EECS 22: Assignment 2

Prepared by: Huan Chen, Prof. Rainer Dömer

October 5, 2016

Due on Thursday 10/20/2016 6:00pm. Note: this is a two-week assignment.

1 Digital Image Processing [100 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, 640×480, 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
640 480
255
RGBRGBRGB...
```

1.2 Initial Setup

Before you start working on the assignment, copy the source code template and image resource file to your own directory by doing the following:

```
mkdir hw2
cd hw2
cp ~eecs22/public/PhotoLab.c .
cp ~eecs22/public/EH.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 file *EH.ppm* is the PPM image 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. 'negative', 'colorfilter', 'edge', 'hflip', 'vmirror', 'border', 'zoom' 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 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:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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: EH
```

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

```
Please make your choice: 1
Please input the file name to load: EH
EH.ppm was read successfully!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
```

```
 10: Test all functions
 11: 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 "EH.ppm.ppm" for reading!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: 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: negative
negative.ppm was saved successfully.
negative.jpg was stored for viewing.
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: 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 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 1: An image and its negative counterpart.

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

```
Please make your choice: 3
"Negative" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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

1.3.4 Color-Filter an image

The functionality of the color filter is to change the selected color in the picture to the color the user wants. To do that, first the user has to choose the color by entering the RGB intensity and the threshold of the color the user wants to modify. Also, the users have to enter the replacement value for each RGB component. All pixels in the picture with color in the chosen range will be replaced with new color. The following shows the pseudo code for the color filter.

```
if (R in the range of [target_r - threshold, target_r + threshold]) and
   (G in the range of [target_g - threshold, target_g + threshold]) and
   (B in the range of [target_b - threshold, target_b + threshold])
  R = replace_r ;
  G = replace_g ;
  B = replace_b ;
else
  keep the current color
```

You need to define and implement the following function to do this DIP. Note that your program should check boundary for the new color values, i.e. the intensity should be in the range of [0, 255].

```
/* color filter  */
void ColorFilter(unsigned char R[WIDTH][HEIGHT],
                 unsigned char G[WIDTH][HEIGHT],
                 unsigned char B[WIDTH][HEIGHT],
                 int target_r, int target_g, int target_b, int threshold,
                 int replace_r, int replace_g, int replace_b);
```

Figure 2 shows an example of this operation. In this example, we change shade of pink to blue color by setting the target_r = 190, target_g = 100, target_b = 150, threshold = 60, replace_r = 0, replace_g = 0, replace_b = 255.

(a) Original image

(b) Color filtered image

Figure 2: An image and its color filtered counterpart.

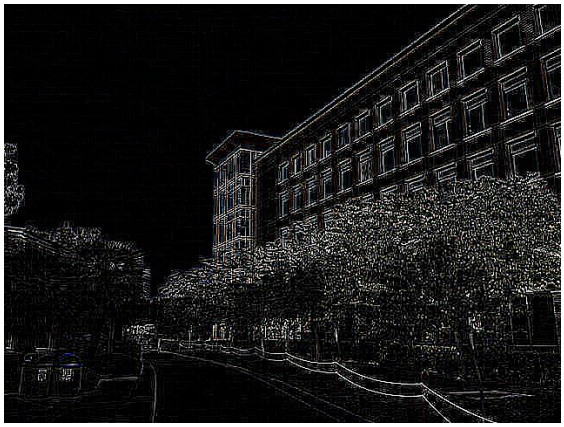
```
Please make your choice: 4
Enter Red   component for the target color: 190
Enter Green component for the target color: 100
Enter Blue  component for the target color: 150
Enter threshold for the color difference: 60
Enter value for Red   component in the target color: 0
Enter value for Green component in the target color: 0
Enter value for Blue  component in the target color: 255
"Color Filter" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
```

```
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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

1.3.5 Edge Detection

(a) Original Image



(b) Edge Image

Figure 3: An image and its edge counterpart.

The edge detection works this way: the intensity value at each pixel is mapped to a new value, which is the sum of itself and its 8 neighbors with different parameters. Note the sum of all parameters is 0, which will result in a vary dark image where only the edges are detected are colored. The following shows an example of the filter and the applied pixel:

```
    Filter             Original Pixels
    X  X  X  X  X       X  X  X  X  X
    X -1 -1 -1  X       X  A  B  C  X
    X -1  8 -1  X       X  D  E  F  X
    X -1 -1 -1  X       X  G  H  I  X
    X  X  X  X  X       X  X  X  X  X
```

To detect an edge of the image, the intensity of the center pixel (E) with the value is changed to $(-A-B-C-D+8*E-F-G-H-I)$. 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 you have to set the boundary for the newly generated pixel value, i.e., the value should be within the range of [0,255].
Hint (the following code can be used to handle pixel intensity under- or overflow):

```
if (v<0) v=0;
else  if (v>255) v=255;
```

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

```
/* Find edge of an image */
void Edge(unsigned char R[WIDTH][HEIGHT],
          unsigned char G[WIDTH][HEIGHT],
          unsigned char B[WIDTH][HEIGHT]);
```

The edge image should look like the figure shown in Figure 3(b):

```
Please enter your choice: 5
"Edge" operation is done!
------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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

1.3.6 Flip Image Horizontally

(a) Original image

(b) Horizontally flipped image

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

```
Please make your choice: 6
"HFlip" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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

1.3.7 Mirror Image Vertically

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

```
                          1 4 6                                 1 4 6
                          5 2 1                                 5 2 1
before vertical mirror:   7 8 9         after vertical mirror: 7 8 9
                          8 2 4                                 5 2 1
                          9 3 7                                 1 4 6
```

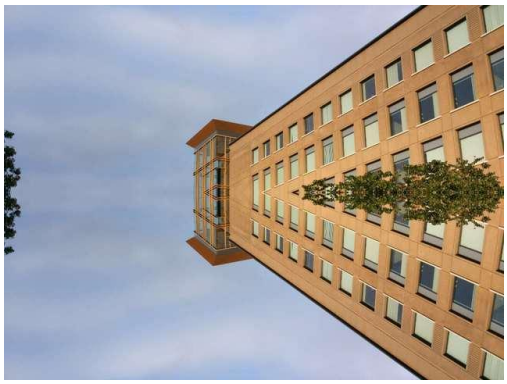
You need to define and implement the following function to do this DIP.

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

Figure 5 shows an example of this operation. Your program's output for this option should be like:

```
Please make your choice: 7
"VMirror" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
```

(a) Original image



(b) Vertically mirrored image

Figure 5: An image and its vertically mirrored counterpart.

```
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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

1.3.8 Add borders to an image

(a) Original image

(b) Image with borders, border color = black, width = 64 pixels

Figure 6: An image and its counterpart when borders are added.

This operation will add borders to the current image. The border color and width (in pixels) of the borders are parameters given by the user. Within your implementation, you should define an aspect ratio of 16:9 for your horizontal to vertical border. This should make your horizontal border look thicker than the vertical one. Figure 6 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],
               char color[SLEN], int border_width);
```

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

```
please make your choice: 8
Enter border width: 64
Available border colors : black, white, red, green, blue, yellow, cyan, pink, orange
Select border color from the options: black
"Border" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

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

1.3.9 Zooming an image

(a) Original image

(b) Zoomed image

Figure 7: An image and its zoomed version counterpart.

For this part of the assignment, you will try to implement a zoom option for the image. This is how it works: You first take two adjacent pixels and subtract the first pixel from the second one. You divide the difference by a zooming factor *(K)* and call it the zooming constant *(C)*. Note that *C* can be positive or negative. For this assignment, we have set *K=2*.

You need to add *C* to the first pixel and create a new pixel between the two original pixels. In theory, you would continue to add *C* to the previous pixel and keep repeating this until you reach the second original pixel, however, in this assignment you will only place one pixel in between two adjacent pixels. Think about how many pixels you would need to place in between two adjacent pixels if *K=3*?

You need to define and implement the following function to do this DIP. The part of the image you want to zoom in is within the area of (0, 0), (320, 0), (320, 210) and (0, 210).

```
/* Zoom an image */
void Zoom(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);
```

The following shows an example: Note that the original image has 2 rows and 5 columns whereas the zoomed version has 3 rows and 9 columns. Do you see a relationship between *K* and size of the zoomed image?

```
      Original                          Zoom (entire)
  10  20  30  40  50          10  15  20  25  30  35  40  45  50
  60  70  80  90  100         35  40  45  50  55  60  65  70  75
                              60  65  70  75  80  85  90  95  100
```

Since, we are only interested in zooming the top left corner of an image with the same size as the orignial one, we only take the pixels of interest:

```
      Zoom (entire)                       Zoom (top left)
10  15  20  25  30  35  40  45  50          10  15  20  25  30
35  40  45  50  55  60  65  70  75          35  40  45  50  55
60  65  70  75  80  85  90  95  100
```

Figure 7 shows an example of this operation. Your program's output for this option should be like:

```
Please make your choice: 9
"Zoom" operation is done!
------------------------------
 1:  Load a PPM image
 2:  Save an image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Color filter an image
 5:  Sketch the edge of an image
 6:  Flip an image horizontally
 7:  Mirror an image vertically
 8:  Add Border to an image
 9:  Zoom an image
 10: Test all functions
 11: Exit
please make your choice:
```

1.3.10 Test all functions

This option helps the user to test all previous functions in one shot. You dont have to implement a test all function; just use the provided one, AutoTest. This function call DIP functions one by one to test and generate the outputs. The function looks like:

```
void AutoTest(unsigned char R[WIDTH][HEIGHT],
              unsigned char G[WIDTH][HEIGHT],
              unsigned char B[WIDTH][HEIGHT])
{
 char fname[SLEN] = "EH";
 char sname[SLEN];

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

ReadImage(fname, R, G, B);
ColorFilter(R, G, B, 190, 100, 150, 60, 0, 0, 255);
strcpy(sname, "colorfilter");
SaveImage(sname, R, G, B);
printf("Color Filter tested!\n\n");

...
...
}
```

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

```
Please make your choice: 10

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

EH.ppm was read successfully!
"Color Filter" operation is done!
colorfilter.ppm was saved successfully.
colorfilter.jpg was stored for viewing.
Color Filter tested!

...
...
```

After runing this fucntion successfully, you are able to see all images in a web browser at: *http://newport.eecs.uci.edu/~youruserid*

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]);

/* save a processed image */
int  SaveImage(char fname[SLEN], 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]);

/* color filter */
void ColorFilter(unsigned char R[WIDTH][HEIGHT],
                 unsigned char G[WIDTH][HEIGHT],
                 unsigned char B[WIDTH][HEIGHT],
                 int target_r, int target_g, int target_b, int threshold,
                 int replace_r, int replace_g, int replace_b);

/* edge detection */
void Edge(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 vertically */
void VMirror(unsigned char R[WIDTH][HEIGHT],
             unsigned char G[WIDTH][HEIGHT],
             unsigned char B[WIDTH][HEIGHT]);

/* add border */
void AddBorder(unsigned char R[WIDTH][HEIGHT],
               unsigned char G[WIDTH][HEIGHT],
               unsigned char B[WIDTH][HEIGHT],
               color[SLEN],
               int border_width);

/* Zoom an image */
void Zoom(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  640 /* Image width */
#define HEIGHT 480 /* image height */
#define SLEN    80 /* maximum length of file names and strings */
#define ZOOM_FACTOR  2 /* Zooming factor for the zoom function */
```

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 $255 - R[][]$ directly back to the 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 'negative', 'colorfilter', 'edge', 'hflip', 'vmirror', 'border', 'zoom' 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.txt*
- *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.

4 Grading

- *Handle menu and user input correctly (20 pts)*
- *DIP operations from menu item 3 to 7, 9 (60 pts, 10 pts each)*
- *Autotest works (10 pts)*
- *Code cleanliness (10 pts), if your code is not well formatted, you will lose partial points*
- *Bonus: menu item 8 add border (10 pts)*