EECS 10: Assignment 7

November 13, 2009

Due on Monday 11/30/2009 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, 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, do the following:

```
cd ~
mkdir -p hw7
cd hw7
cp ~eecs10/hw7/PhotoLab.c .
cp ~eecs10/hw7/pumpkins.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 *pumpkins.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/∼userid* (If you access the server out of campus, use the link *http://newport.eecs.uci.edu/∼userid/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 the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  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: pumpkins
```

After a name, for example *pumpkins*, is entered, the *PhotoLab* will load the file pumpkins.ppm. If it is read correctly, the following is shown:

```
please make your choice: 1
Please input the file name to load: pumpkins
pumpkins.ppm was read successfully!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
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 "pumpkins.ppm.ppm" for reading!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
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 enter 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 the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
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/∼userid* (For off campus, the link is: *http://newport.eecs.uci.edu/∼userid/imagename.jpg*)

(a) Color image

(b) Black and white image

Figure 1: A color image and its bw counterpart.

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. Figure 1 shows an example of this operation.

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

```
please make your choice: 3
"Black & White" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
```

(a) Original image

(b) Night view image

Figure 2: An image and its night view.

```
please make your choice:
```

1.3.4 Apply Night View Effect to the Image

Night View effect only keeps the green channel intensity of the image. To achieve this, red and blue intensity values at a pixel are set to 0, and the result is assigned to the pixel as a new intensity. You need to define and implement the following function to do this DIP.

```
/* Night View in the photo */
void NightView(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
"Night View" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
```

(a) Original image

(b) Horizontally mirrored image

Figure 3: An image and its horizontally mirrored counterpart.

```
13:  Exit
please make your choice:
```

1.3.5 Mirror Image Horizontally

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

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

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

```
please make your choice: 5
"HMirror" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
```

(a) Original image

(b) Horizontally flipped image

Figure 4: An image and its horizontally flipped counterpart.

```
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

1.3.6 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
```

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 the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

(a) Original image

(b) Vertically flipped image

Figure 5: An image and its vertically flipped counterpart.

1.3.7 Flip Image Vertically

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

```
                        1 0 5                          5 4 9
                        2 1 6                          4 3 8
  before vertical flip: 3 2 7    after vertical flip:  3 2 7
                        4 3 8                          2 1 6
                        5 4 9                          1 0 5
```

You need to define and implement the following function to do this DIP. Figure 5 shows an example of this operation.

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

```
please make your choice: 7
"Vertical Flip" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
```

```
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

(a) Original image

(b) 4x Zoom-In image

Figure 6: An image and its 4x Zoom-In counterpart

1.3.8 nX Zoom-In

In this section, you need to implement a zoom-in nx function from a point in the image as zoom-in center, that enlarges the image area by a factor of n, where n should be a small integer (2, 4, 8 ...).
To zoom in, the intensity values in the center of the picture are distributed to the whole picture. The following shows an example of 2x with zoom in center in the center of the image:

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

Gerenally, for "magnitude"X zoom in at center (centerX, centerY), pixel (x, y) should copy R,G,B intensity from pixel (offsetX, offsetY), where

$$offsetX = (x - WIDTH/2)/magnitude + centerX$$

$$offsetY = (y - HEIGHT/2)/magnitude + centerY$$

You have to check if (offsetX, offsetY) is within the matrix ($0 <= offsetX < WIDTH, 0 <= offsetY < HEIGHT$).
You need to define and implement the following function to do this DIP. (centerX, centerY) is the zoom in center. "magnitude" is the zoom in factor (2x 4x 8x ...). You program should ask the user to input centerX, centerY and magnitude. In the script use (161, 298) as zoom center and 4 as magnitude. This will zoom in the pumpin in the image.

```
/* Zoom in the photo */
void Zoomin(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
```

```
unsigned char B[WIDTH][HEIGHT],
int centerX, int centerY, int magnitude);
```

Figure 6 shows an example of this operation.

```
please make your choice: 8
Please input x coordinate of the zoom center: 161
Please input y coordinate of the zoom center: 298
Please zoom magnitude: 4
"Zoomin" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

-

(a) Watermark image

(b) Watermark image over original image

Figure 7: Watermark image and result of watermark operation

1.3.9 Watermark

This function increases the color intensity of certain pixels in the original image according to pixel information on a watermark image. In our program, we will use an anteater image (*anteater.ppm*) as the watermark image(see Figure7(a)).

The size of *anteater.ppm* is much smaller than the original image, which is 249 by 100 pixels. It is watermarked on the original image in a tiled fashion as Figure7(b) shows, which means the anteater image is repeatly watermarked on orignial one vertically and horizontally until the the orignial image is filled with water mark.

To achieve the watermark effect, the white background in the watermark image is considered as transparent. That is, for each the non-white pixel in anteater.ppm, the R,G and B intensity of the corresponding pixel on the orginal image is increased by 50, which highlights the corresponding pixels on the original image. Whether or not a pixel in *anteater.ppm* is a white pixel can be determined by the RGB values of this pixel: if the RGB values of a pixel are 255/255/255, it is considered to be a white pixel.

You need to define and implement the following function to do this DIP. The first parameter fname is the file name of the watermark image (*watermark.ppm*). The watermark image should be read with ReadImageW() instead of ReadImage(), since the watermark image has a different image size. Three two dimensional arrays should be declared to hold the R,G,B intensities of watermark image just as the orginal image.

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

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

```
please make your choice: 9
Please input the file name for the second image: anteater
anteater.ppm was read successfully!
"Watermark" operation is done!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

After the successful watermark operation, the watermarkeded image should like the figure shown in Figure 7(b):

(a) Original image

(b) Blured image

Figure 8: An image and its blur counterpart.

1.3.10 Blur (bonus points: 10pt)

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.

```
/* 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 aged 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 the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

(a) Original image

(b) Timestamp removed

Figure 9: An image and its timestamp removal counterpart.

1.3.11 Copy Region (bonus points: 10pt)

Copy region function copies a w*h region from orginal top left point (x1, y1) to new top left point(x2, y2). All pixels in the region constrained by (x1, y1) (top left pixel) and (x1 + w - 1, y1 + h - 1) (bottom right pixel) are copied to the region constrained by (x2, y2) (top left pixel) and (x2 + w - 1, y2 + h - 1) (bottom right pixel).

With region copy, we can copy a region near the timestamp on the photo to the timestamp region. This way the timestamp can be overlayed. In this assignment, we ask you to copy a 95*20 region from (468, 405) to (468, 425), which exactly removes the timestamp on the pumpkins image.

```
/* copy a region on the photo */
void CopyRegion(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT], unsigned int x1, unsigned int y1,
unsigned int x2, unsigned int y2, unsigned int w, unsigned int h);
```

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

```
please make your choice: 11
Please input x coordinate of source: 468
Please input y coordinate of source: 405
Please input x coordinate of destination: 468
Please input y coordinate of destination: 425
Please input width of the region: 95
Please input height of the region: 20
"Copy Region" operation is done!
--------------------------------
 1:  Load a PPM image
```

```
 2:  Save the image in PPM and JPEG format
 3:  Change a color image to black and white
 4:  Change the image to night view
 5:  Mirror the image horizontally
 6:  Flip the image horizontally
 7:  Flip the image vertically
 8:  Zoom in
 9:  Add watermark to the image
10:  Blur the image (10 extra points)
11:  Copy region (10 extra points)
12:  Test Automation
13:  Exit
please make your choice:
```

1.3.12 Test automatioin

We provided a test automation function (BatchTest()) for you to test all functionalities (just like what you would do manually when preparing the PhotoLab.script) in your program in one shot. It is for *testing* and *grading* purpose only. It is not designed for debugging. We need your implementation of menu item "Test Automation" to grade your homework. In your swich statement, add the following code to call BatchTest()

```
        case 12:
            printf("Test automation for all functionalities:\n");
            BatchTest(R, G, B);
            return 0;
```

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

```
please make your choice: 12
Test automation for all functionalities:

pumpkins.ppm was read successfully!
bw.ppm was saved successfully.
bw.jpg was stored for viewing.
"Black & White" operation has been tested!

pumpkins.ppm was read successfully!
night.ppm was saved successfully.
night.jpg was stored for viewing.
"Night View" operation has been tested!

pumpkins.ppm was read successfully!
hmirror.ppm was saved successfully.
hmirror.jpg was stored for viewing.
"HMirror" operation has been tested!

pumpkins.ppm was read successfully!
hflip.ppm was saved successfully.
hflip.jpg was stored for viewing.
```

```
"HFlip" operation has been tested!

pumpkins.ppm was read successfully!
vflip.ppm was saved successfully.
vflip.jpg was stored for viewing.
"VFlip" operation has been tested!

pumpkins.ppm was read successfully!
zoomin.ppm was saved successfully.
zoomin.jpg was stored for viewing.
"Zoomin" operation has been tested!

pumpkins.ppm was read successfully!
anteater.ppm was read successfully!
watermark.ppm was saved successfully.
watermark.jpg was stored for viewing.
"Watermark" operation has been tested!

pumpkins.ppm was read successfully!
blur.ppm was saved successfully.
blur.jpg was stored for viewing.
"Blur" operation has been tested!

pumpkins.ppm was read successfully!
copy.ppm was saved successfully.
copy.jpg was stored for viewing.
"Copy Region" operation has been 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 watermark image from a file */
int ReadImageW(char fname[SLEN], unsigned char R[WWIDTH][WHEIGHT],
unsigned char G[WWIDTH][WHEIGHT], unsigned char B[WWIDTH][WHEIGHT]);

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

/* Night Vision in the photo */
void NightView(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]);

/* 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]);

/* Zoom in the photo */
void Zoomin(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT], int centerX, int centerY, int magnitude);

/* copy a region on the photo */
void CopyRegion(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT], unsigned int x1, unsigned int y1,
```

```
unsigned int x2, unsigned int y2, unsigned int w, unsigned int h);

/* blur the photo */
void Blur(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 file name length */
#define WWIDTH 249   /* watermark image width */
#define WHEIGHT 100  /* watermark 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 NightView() 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. Perform the following operations
   - Load *pumpkins.ppm*
   - Convert the current image to black and white and save it as *bw*
   - Reload the original picture *pumpkins.ppm*
   - Convert the current image to night view and save it as *night*
   - Reload the original picture *pumpkins.ppm*
   - Mirror the current image horizontally and save it as *hmirror*
   - Reload the original picture *pumpkins.ppm*
   - Flip the current image horizontally and save it as *hflip*
   - Reload the original picture *pumpkins.ppm*
   - Flip the current image vertically and save it as *vflip*
   - Reload the original picture *pumpkins.ppm*
   - Zoom in the current image 4X in center(161, 298) and save it as *zoomin*
   - Reload the original picture *pumpkins.ppm*
   - Watermark the image with *anteater* and save it as *watermark*

   For the bonus parts only:

   - Reload the original picture *pumpkins.ppm*
   - Blur the current image and save it as *blur*
   - Reload the original picture *pumpkins.ppm*
   - Copy a 95*20 region from (468, 405) to (468, 425) in current image and save it as *copy*(only if you do the bonus points
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.