EECS 22: Assignment 3

Prepared by: Nistha Tandiya, Prof. Rainer Doemer

October 23, 2014

Due on Monday 11/11/2014 11:00pm. Note: this is a two-week assignment.

1 Digital Image Processing [100 points + 10 bonus points]

In this assignment you will practice how to break a program into multiple modules and compile them into one program. Based on the program *PhotoLab* for Assignment 2, you will be asked to develop some advanced digital image processing (DIP) operations, partition them in separate modules, manipulate images using bit operations, and develop an appropriate **Makefile** to compile your program with DEBUG mode on or off.

1.1 Introduction

In Assignment 2, you were asked to develop an image manipulation program *PhotoLab* by using DIP techniques. 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 by using the *PhotoLab*. This assignment will be based on Assignment 2.

1.2 Initial Setup

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

```
mkdir hw3
cd hw3
cp -r /users/grad2/doemer/eecs22/hw3/Spider.ppm .
cp -r /users/grad2/doemer/eecs22/hw3/Peter.ppm  .
cp -r /users/grad2/doemer/eecs22/hw3/index.html .
```

We will extend the *PhotoLab* program based on Assignment 2. Please reuse your **PhotoLab.c** file as the starting point for this assignment. You may need to copy **PhotoLab.c** from the *hw2* folder to *hw3* first.

We will use the PPM image file *RingMall.ppm* for this assignment. Once a DIP operation is done, you can save the modified image as *name.ppm*, and it 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 any web browser at: *http://newport.eecs.uci.edu/∼youruserid*, if required names are used. 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 Decompose the program into multiple modules

Please decompose the **PhotoLab.c** file into multiple modules and header files, as follows:

- **PhotoLab.c**: the main module contains the *main()* function, and the menu function *PrintMenu()* as well as *AutoTest()*.

- **FileIO.c**: the module for the function definitions of *ReadImage()* and *SaveImage()*.
- **FileIO.h**: the header file for **FileIO.c**, with the function declarations of *ReadImage()* and *SaveImage()*.

  Note: In SaveImage() function, you will have to change the path when the ppm file is changed to jpg

  ```
  sprintf(SysCmd, "/users/grad2/doemer/eecs22/bin/pnmtojpeg_hw3.tcsh %s",
  fname_tmp2);
  ```

- **Constants.h**: the header file in which the constants to be used are defined.
- **DIPs.c**: the module for the DIP function definitions in Assignment 2, i.e. *BlackNWhite()*, *VFlip()*, *HMirror()*, *ColorFilter()*, *Edge()*, *Shuffle()*.
- **DIPs.h**: the header file for **DIPs.c**, with the corresponding DIP function declarations.
- **Advanced.c**: the module for the function definitions of new filters in Assignment 3, *Posterize()*, *FillLight()*, *Overlay()* and *CutPaste()*
- **Advanced.h**: the header file for **Advanced.c**, with the function declarations of *Posterize()*, *FillLight()*, *Overlay()* *and* CutPaste() .

HINT: Please refer to the slides of *Lecture 7* for an example of decomposing programs into different modules.

1.4 Compile the program with multiple modules using a static shared library

The *PhotoLab* program is now modularized into different modules: **PhotoLab**, **FileIO**, **DIPs** and **Advanced**. In this assignment we are using shared libraries to group the compiled object code files in to static libraries. Typically C functions and methods which can be shared by more than one application are broken out of the application's source code, compiled and bundled into a library.
As shown in Lecture 7, to generate the libraries first compile the source code. Use *"-c"* option for **gcc** to generate the object files for each module, e.g.

```
% gcc -c FileIO.c -o FileIO.o -ansi -Wall
% gcc -c DIPs.c -o DIPs.o -ansi -Wall
% gcc -c Advanced.c -o Advanced.o -ansi -Wall
...
```

Libraries are typically names with the prefix "lib". Here we want to create two libraires, *libFileIO.a* and *libFilter*:

```
% ar rc libFileIO.a FileIO.o
% ranlib libFileIO.a
% ar rc libFilter.a DIPs.o Advanced.o
% ranlib libFilter.a
```

Linking with the library:

```
% gcc PhotoLab.c -lFileIO -lFilter -L. -o PhotoLab
```

Execute the program:

```
% ./PhotoLab
program executes
% _
```

1.5 Using 'make' and 'Makefile'

To simplify the build process, we can put the commands above into a **Makefile** and use the *make* utility to automatically build the executable program from source code. Please create your own **Makefile** with at least the following targets:

- *all*: the target to generate the executable programs.
- *clean*: the target to clean all the intermediate files, e.g. object files, libraries, and the executable program(s).
- *PhotoLabTest*: executable target to run PhotoLabTest (see below).
- *PhotoLab*: executable target to generate the interactive program *PhotoLab*.

To use your **Makefile**, please use this command:
`% make all`
The executable programs *PhotoLab* and *PhotoLabTest* shall then be automatically generated.

HINT: Please refer to the slides of *Lecture 8* for an example on how to create a **Makefile**.

1.6 Advanced DIP operations

In this assignment, please add one more module named **Advanced**, consisting of **Advanced.c** and **Advanced.h** and implement the advanced DIP operations described below.

Please reuse the menu you designed for Assignment 2 and extend it with the advanced operations. 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 & White
  4: Flip an image vertically
  5: Mirror an image horizontally
  6: Color-Filter an image
  7: Sketch the edge of an image
  8: Shuffle an image
  9: Posterize the image
 10: Fill Lights to an image
 11: Overlay an image
 12: Bonus, Cut and Paste operation on image
 13: Test all functions
 14: Exit
```

Note: Option 12: 'Cut and Paste operation' is a bonus task (10pts).

1.6.1 Bit Manipulations: Posterize the image

Posterization of an image refers to conversion of a continuous gradation of tone to several regions of fewer tones, with abrupt changes from one tone to another. The term posterization is used because it can influence your photo similar to how the colors may look in a mass-produced poster, where the print process uses a limited number of color inks. (`http://en.wikipedia.org/wiki/Posterization`).

We are going to use bit manipulations to posterize the image.

- **Posterize the image**
  As before, a pixel in the image is represented by a 3-tuple (R, G, B) where R, G, and B are the values for the intensities of the red, green, and blue channels respectively.
  The range of R, G, and B are from 0 to 255 inclusively. As such, we use *unsigned char* variables to store the values of these three values.

To posterize the image, we are going to change the least significant n, $\{n \in (1,2,3,\ldots 8)\}$ bits of color intensity values so as to change the tone of the pixels. Basically, we will change the nth least significant bit of the color intensity value to be 0, and the least $n-1$ bits to be all 1.

For example, assume that the color tuple of the pixel at coordinate(0,0) is (41, 84, 163). Therefore,

```
R[0][0] = 41
G[0][0] = 84
B[0][0] = 163
```

In binary representation, the color tuple will be:

$$\texttt{R[0][0]} = 00101001_2$$
$$\texttt{G[0][0]} = 01010100_2$$
$$\texttt{B[0][0]} = 10100011_2$$

Fig. 1 shows the operation of posterization for different least significant bits of the intensities for the red, green, and blue channels. As illustrated in Fig. 1(a), in order to posterize the least 6 significant bits of the red intensity, we set the 6th bit to be 0, and the 1st to the 5th bits to be 1s.
Similarly in Fig. 1(b), to posterize the least 5 significant bits of the green intensity, we set the 5th bit to be 0, and the 1st to the 4th bits to be 1s; and in Fig. 1(c), to posterize the least 4 significant bits of the blue intensity, we set the 4th bit to be 0, and the 1st to the 3th bits to be 1s.

(a) Posterize the least 6 significant bits of the red channel for pixel(0,0)

(b) Posterize the least 5 significant bits of the green channel for pixel(0,0)

(c) Posterize the least 4 significant bits of the blue channel for pixel(0,0)

Figure 1: Example of posterizing the color channels.

- **Function Prototype:** You need to define and implement the following function to do this DIP.

```
/* Posterize the image */
void Posterize(unsigned char R[WIDTH][HEIGHT],
               unsigned char G[WIDTH][HEIGHT],
               unsigned char B[WIDTH][HEIGHT],
               unsigned int pbits);
```

In this assignment, we will input just a single value $(pbits)$ from the user for the least significant posterizing bits for all three colors.Since the size of *unsigned char* variable is 8 bits, the valid range of *pbits* will be $[1,8]$.

HINT: You will need to use bitwise operators, e.g. `&`, `<<`, `>>`, `|` for this operation.

(a) Image without posterization

(b) Image with posterization, where pbits = 7

Figure 2: The image and its posterized counterpart.

Fig. 2 shows an example of our posterized image.

When the user chooses the posterization option, the output should look like:

```
please make your choice: 9
Enter the number of posterization bits (1 to 8): 7
"Posterize" operation is done!
-------------------------------
  1: Load a PPM image
  2: Save an image in PPM and JPEG format
  3: Change a color image to Black & White
  4: Flip an image vertically
  5: Mirror an image horizontally
  6: Color-Filter an image
  7: Sketch the edge of an image
  8: Shuffle an image
  9: Posterize the image
 10: Fill Lights to an image
 11: Overlay an image
 12: Bonus, Cut and Paste operation on image
 13: Test all functions
 14: Exit
```

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

1.6.2 Fill the image with Christmas lights

In this operation, you have to add colorful Christmas lights to an image. You need to define and implement a function to do the job. Your function should input two parameters from the user: one for the number of lights and second for the width of each light. The function should be able to randomly place the lights on the image. Further, the color of lights should be randomly picked to be red $(255, ,0\,0)$, blue $(0\,,255,\,0)$ or green $(0,\,0,\,255)$. You need the knowledge of the random number generator which was used in Assignment 1.

The structure of each light is detailed in Figure 3

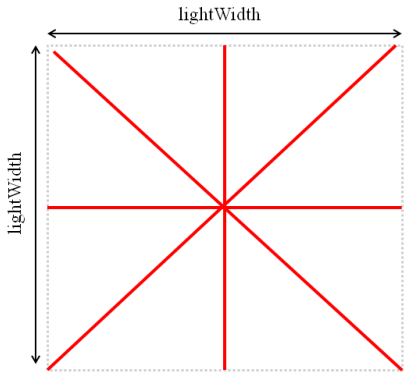


Figure 3: Structure of the light

Caution: The positions of the center of the lights should be such that the entire light fits in the image area. Failure to do so might give you a segmentation fault. To prevent segmentation fault, ensure that the center pixel for light is positioned in the following range :

$$x \in \left(\frac{lightWidth}{2},\ WIDTH - \frac{lightWidth}{2}\right)$$

$$y \in \left(\frac{lightWidth}{2},\ HEIGHT - \frac{lightWidth}{2}\right)$$

Figure 4 shows an example of this operation with 200 lights, each set to width 20.

(a) Original image

(b) Image with Light

Figure 4: An image before and after FillLight operation.

- **Function Prototype:** You need to define and implement the following function to do this DIP.

```
/* Fill christmas lights to image */
void FillLight(  int number, int lightWidth
unsigned char R[WIDTH][HEIGHT],
unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);
```

Here, *number* specifies the number of lights desired in the image. *lightWidth* gives the dimension of each light.

Figure 4 shows an example of this operation. Once the user chooses this option, your program's output should like this:

```
Please make enter your choice: 10
Please input number of lights: 200
Please input the width of each light: 20
"FillLight" operation is done!
--------------------------------
  1: Load a PPM image
  2: Save an image in PPM and JPEG format
  3: Change a color image to Black & White
  4: Flip an image vertically
  5: Mirror an image horizontally
  6: Color-Filter an image
  7: Sketch the edge of an image
  8: Shuffle an image
  9: Posterize the image
 10: Fill Lights to an image
 11: Overlay an image
 12: Bonus, Cut and Paste operation on image
 13: Test all functions
 14: Exit
please enter your choice:
```

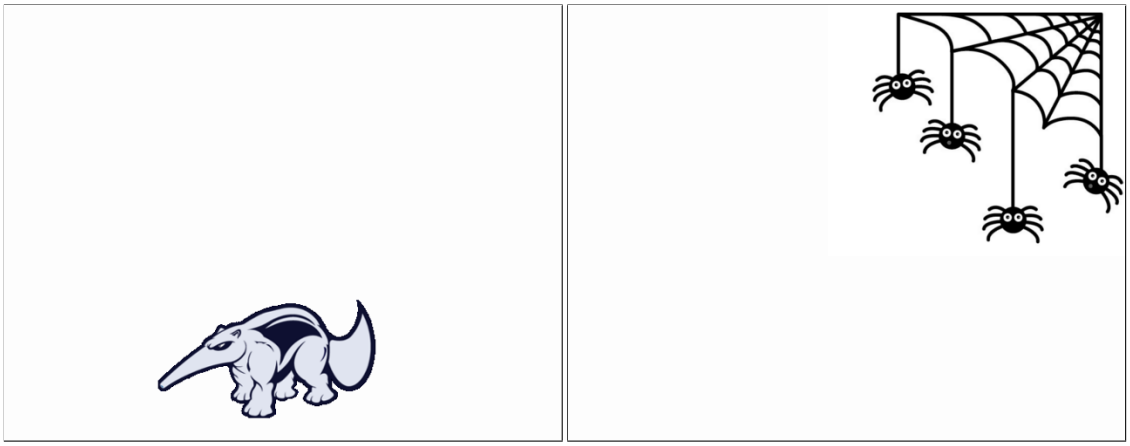
Save the image with name 'light' after this step.

1.6.3 Image Overlay

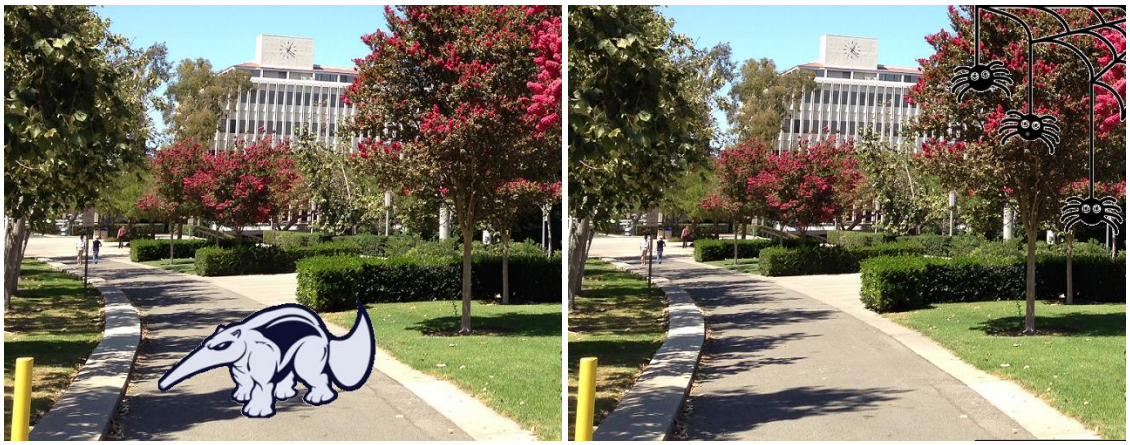
This function overlays the current image with a second image. In our program, we will put an image of our UCI mascot Peter and an image of halloween spiders on the original image.

To start the implementation, you need to prompt 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 (*ReadImage()* for loading). In this assignment, the second image is *Peter.ppm* and *Spider.ppm*, with dimensions 640×500 pixels (same size as original image). The user also needs to enter the position of the overlay with coordinates (x, y).

Take a look at Fig. 5(a) and Fig. 5(b) images. Note that they have white backgrounds which are inconsistent with our original image. To achieve the overlay effect, we will treat the white background as transparent color. That is, each of the non-background pixels will be overlaid to a position in the original image, whereas background pixels will stay as in the original image. Whether a pixel in images (a) through (b) is a background pixel or not can be decided by the RGB values of this pixel. More specifically, if a pixel has RGB value greater than (250, 250, 250) i.e. $(r \geq 250 \quad and \quad g \geq 250 \quad and \quad b \geq 250\ )$ which represents white background, this pixel should not be put onto the original image.



(a) Peter image (b) Halloween image



(c) Overlay the peter image at position (0,0) (d) Overlay the halloween spider image at position (90,-10)

Figure 5: An image and the overlayed image.

Caution: Pay attention to the coordinates after applying the offset values! Any pixel outside of the valid ranges of the original image shoud be ignored (otherwise segmentation faults may occur).

- **Function Prototype:** You need to define and implement the following function to do this DIP.

```
/*Overaly an image onto the original 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);
```

Here, x_offset, y_offset are the overlay coordinates.

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

```
Please make your choice: 11
Please input the file name for the second image: Peter
```

```
Please input x coordinate of the overlay image: 0
Please input y coordinate of the overlay image: 0
Peter.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 & White
  4: Flip an image vertically
  5: Mirror an image horizontally
  6: Color-Filter an image
  7: Sketch the edge of an image
  8: Shuffle an image
  9: Posterize the image
 10: Fill Lights to an image
 11: Overlay an image
 12: Bonus, Cut and Paste operation on image
 13: Test all functions
 14: Exit
Please make your choice: 11
Please input the file name for the second image: Spider
Please input x coordinate of the overlay image: 90
Please input y coordinate of the overlay image: -10
Spider.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 & White
  4: Flip an image vertically
  5: Mirror an image horizontally
  6: Color-Filter an image
  7: Sketch the edge of an image
  8: Shuffle an image
  9: Posterize the image
 10: Fill Lights to an image
 11: Overlay an image
 12: Bonus, Cut and Paste operation on image
 13: Test all functions
 14: Exit
Please make your choice:
```

The effect can be seen in Figure 5. In Fig. 5(c), the position chosen is (0,0) for the Peter's image, and in Fig. 5(d) (90,-10) for the Halloween spider's image.
Save the images with names "overlay_peter" and "overlay_spider" after this step.

1.7 BONUS: Cut Paste Function (10 points)

This is a bonus task. In this part, you have to implement a function to perform a cut and paste operation on the image. The function should take the dimensions of the block which should be used for pasting and the number of total pastes required. The (x, y) coordinates for the paste locations should be taken by the function when it executes.

Figure 6 shows the result of the operation with three paste points. The block for pasting was taken from the coordinates $(370, 20)$ and had dimension 27×57. You can see that the clock wall in the building has vanished along

(a) Original image

(b) Image after Cut Paste

Figure 6: An image before and after Cut Paste operation.

with its shadow.

- **Function Prototype:** You need to define and implement the following function to do this DIP.

```
/* Perform Cut Paste operations on the image*/
void CutPaste( unsigned char R[WIDTH][HEIGHT],
       unsigned char G[WIDTH][HEIGHT],
       unsigned char B[WIDTH][HEIGHT],
       unsigned int startX,
       unsigned int startY,
       unsigned int x_width,
           unsigned int y_width, unsigned int pasteNumber);
```

Here *startX*, *startY*, *x_width*, *y_width* represent the dimension of the block from the original image which needs to be pasted on other points. *pasteNumber* give the number of points where the block needs to be pasted. The coordinates of the places where paste operation has to be done is inputed during the execution of the function.
Once user chooses this option, your program's output should be like:

```
Please make your choice: 12
Give the start X coordinate for cutting: 370
Give the start Y coordinate for cutting: 20
Give the width for cutting: 27
Give the height for cutting: 57
Give the number of paste locations: 3
Please input start x and y coordinate for Paste no 1 : 341 19
Please input start x and y coordinate for Paste no 2 : 315 18
Please input start x and y coordinate for Paste no 2 : 288 16
Cut Paste Operation Completed !
--------------------------------
  1: Load a PPM image
  2: Save an image in PPM and JPEG format
  3: Change a color image to Black & White
  4: Flip an image vertically
  5: Mirror an image horizontally
```

```
  6: Color-Filter an image
  7: Sketch the edge of an image
  8: Shuffle an image
  9: Posterize the image
 10: Fill Lights to an image
 11: Overlay an image
 12: Bonus, Cut and Paste operation on image
 13: Test all functions
 14: Exit
please enter your choice:
```

Save the image as *cutPaste* after this step

1.8 Test all functions

Finally, you are going to complete the *AutoTest()* function to test all the functions. In this function, you are going to call DIP functions one by one and save 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] = "RingMall";
 char sname[SLEN];

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

 ...

 ReadImage(fname, R, G, B);
 Posterize(R, G, B, 7);
 SaveImage("poster", R, G, B);
 printf("Posterize tested!\n\n");

 ...
 ReadImage(fname, R, G, B);
 Edge(R, G, B);
 SaveImage("edge", R, G, B);
 printf("Edge Detection tested!\n\n");

 ...
}
```

Please use the following arguments for the functions:

- Posterizing Image: 7 as the posterising bits for the red, green, and blue channels.
- Fill Image with lights: 200 as the number of lights with light width 20
- Image Overlay: Peter.ppm with offsets (0, 0) and Spider.ppm with (90, -10) offset

- Cut Paste Operation: Use the coordinates given in the function desription. A correct image would have no clock wall on the administration building

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

```
Please make your choice: 13

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

...

RingMall.ppm was read successfully!
poster.ppm was saved successfully.
poster.jpg was stored for viewing.
Posterize tested!

...

RingMall.ppm was read successfully!
edge.ppm was saved successfully.
edge.jpg was stored for viewing.
EdgeDetection tested!
```

Please implement the *AutoTest()* function in **Photolab.c**. Since the *AutoTest()* function will call the functions in the **DIPs.c** and **Advanced.c** modules, please include the needed header files properly. Also, be sure to adjust your **Makefile** for proper dependencies.

1.9 Support for the DEBUG mode

In C program, *macros* can be defined as preprocessing directives. Please define a macro named **"DEBUG"** in your source code to enable / disable the messages shown in the *AutoTest()* function.

When the macro is defined, the *AutoTest()* function will output as in Section 1.8. Otherwise, there should be no output shown on the screen.
Please decide in which function and in which module this **"DEBUG"** macro needs to be added.

1.10 Extend the Makefile

For the **Makefile**, please

- extend it properly with the targets for your program with the new module: **Advanced.c**.
- generate two executable programs

  1. *PhotoLab* with the user interactive menu and the **DEBUG** mode off.
  2. *PhotoLabTest* without the user menu, but with only the *AutoTest()* function for testing, and turn the **DEBUG** mode on. Note that we can thus use the same source files to generate two different programs.

  Define two targets to generate these two programs. Please use the *"-D"* option for gcc to enable / disable the DEBUG mode instead of defining the "DEBUG" macro in the source code. You may need to define more targets to generate the object files with different DEBUG modes.

2 Implementation Details

2.1 Function Prototypes

For this assignment, you need to define the following functions in **Advanced.h**:

```
/*** function declarations ***/


/*Posterize  Image*/
void Posterize( unsigned char R[WIDTH][HEIGHT],
   unsigned char G[WIDTH][HEIGHT],
   unsigned char B[WIDTH][HEIGHT],
   unsigned char pbits);

/* Add noise to image */
void FillLight( int percentage, int lightWidth,
  unsigned char R[WIDTH][HEIGHT],
  unsigned char G[WIDTH][HEIGHT],
  unsigned char B[WIDTH][HEIGHT]);

/* Overlay with another 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);

/* Perform Cut Paste operations on the image*/
void CutPaste( unsigned char R[WIDTH][HEIGHT],
       unsigned char G[WIDTH][HEIGHT],
       unsigned char B[WIDTH][HEIGHT],
       unsigned int startX,
       unsigned int startY,
       unsigned int x_width,
       unsigned int y_width, unsigned int pasteNumber);
```

You may want to define other functions as needed.

2.2 Global constants

The following global constants should be defined in **Constants.h**(please don't change their names):

```
  #define WIDTH  640   /* Image width */
  #define HEIGHT 500   /* image height */
  #define SLEN    80   /* maximum length of file names */
  #define SHUFF_HEIGHT_DIV 4  /* Height division for shuffling
  #define SHUFF_WIDTH_DIV 4   /* Width division for shuffling
```

Please make sure that you properly include this header file when necessary.

3 Budgeting your time

You have two weeks to complete this assignment, but we encourage you to get started early as there is a little more work than for Assignment 2. We suggest you budget your time as follows:

- Week 1:
  1. Decompose the program into different modules, i.e. **PhotoLab.c**, **FileIO.c**, **FileIO.h**, **Constants.h**, **DIPs.c**, **DIPs.h**.
  2. Create your own **Makefile** and use it to compile the program.
  3. Create module **Advanced.c**, **Advanced.h**, and implement an initial advanced DIP function.
- Week 2:
  1. Implement all the advanced DIP functions.
  2. Implement the *AutoTest()* function.
  3. Implement the **Test.c** file.
  4. Figure out how to enable/disable the **DEBUG** mode in the source code and add targets to the **Makefile** accordingly.
  5. Script the result of your programs and submit your work.

4 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 *PhotoLab* by using your **Makefile**.
3. Choose 'Test all functions' (The file names must be 'bw', ..., 'poster', 'edge' for the corresponding function).
4. Exit the PhotoLab.
5. Compile and run *Test* by using your **Makefile**.
6. Clean all the object files and executable programs by using your **Makefile**.
7. Stop the script by typing the command: *exit*.
8. Rename the script file to *PhotoLab.script*.

NOTE: make sure to 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. ***Please don't open any text editor while scripting !!!***

5 Submission

Use the standard submission procedure to submit the following files as the whole package of your program:

- *PhotoLab.c*
- *PhotoLab.script*
- *FileIO.c*
- *FileIO.h*

- *Constants.h*
- *DIPs.c*
- *DIPs.h*
- *Advanced.c*
- *Advanced.h*
- *Makefile*

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.