EECS 22: Assignment 5

Prepared by: Guantao Liu, Prof. Rainer Dömer

November 16, 2016

Due Thursday December 1, 2016 at 6:00 pm

Contents

1 MovieLab (100 points) **1**
1.1 Introduction . . . 1
1.2 Initial Setup . . . 2
1.3 Design the MovieLab Program . . . 3
1.3.1 The Image.c Module (Provided) . . . 3
1.3.2 The ImageList.c Module . . . 5
1.3.3 The Movie.c Module . . . 6
1.3.4 The MovieLab.c Module . . . 8

2 Bonus (10 points) **14**

3 Build the Makefile **14**

4 Implementation Details **14**
4.1 Structure Definitions . . . 14
4.2 Function Prototypes . . . 15

5 Budgeting Your Time **16**

6 Script File **17**

7 Submission **17**

8 Grading **18**

1 MovieLab (100 points)

In this assignment you will learn how to design a program to take command-line arguments and how to design a linked list. A program, *MovieLab*, will be developed to perform digital image processing (DIP) operations on an input movie. A movie is basically a sequence of images called frames with the same size. You will be asked to design a linked list of images to represent the movie in your program, load the frames of the movie, and then use the DIP functions designed in the previous assignments to perform image processing operations on the frames in the movie.

1.1 Introduction

A movie is basically a sequence of images with different contents but same fixed size. Playing a movie is actually showing the images one after another at a certain rate, i.e. *fps* (frames per second). Each image in the movie is the same as what we have learned in the previous assignments. It is essentially a two-dimensional matrix, which can be represented in C by an array of pixels. A pixel is still the smallest unit of an image.

In this assignment, we will work on a movie with a fixed number of frames (180) and resolution (320×240 pixels/frame), but your program should be able to handle other sizes as well. The color space of the images in the movie is **YUV** format (`http://en.wikipedia.org/wiki/YUV`) instead of **RGB**.

In **YUV** format, the color of each pixel is still represented by 3 components, now referred to as *Y channel*, *U channel* and *V channel*. Here, *Y channel* represents the luminance of the color, while *U channel* and *V channel* represent the chrominance of the color. Each channel for one pixel is still represented by an intensity value between 0 and 255. In order to utilize the DIP functions that handle the images using the **RGB** color space, conversion is needed to change the **YUV** tuple into a **RGB** tuple for each pixel (Section 1.3.3). The **YUV** color space is very common for video streams. As our input and output file both use the **YUV** color space, we need to convert each frame in the movie from **YUV** to **RGB** right after loading the movie, and from **RGB** to **YUV** before saving the movie to the output file.

1.2 Initial Setup

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

1. Create the directory *hw5* for this assignment, and change your current directory to *hw5* by using these commands:

   ```
   mkdir hw5
   cd hw5
   ```

2. We will reuse some of the source code from our previous assignments. Please feel free to reuse any of your designs, or reuse the solution files to the previous assignments which are posted on our course website. Please integrate the necessary DIP functions defined in the DIPs.c and Advanced.c from Assignment 4 into a new DIPs.c for Assignment 5.

   Note: Instead of the course website, you can copy the solutions to Assignment 4 from the shared folder:

   ```
   cp ~eecs22/public/hw5/FileIO.h .
   cp ~eecs22/public/hw5/FileIO.c .
   cp ~eecs22/public/hw5/DIPs.h .
   cp ~eecs22/public/hw5/DIPs.c .
   cp ~eecs22/public/hw5/Advanced.h .
   cp ~eecs22/public/hw5/Advanced.c .
   cp ~eecs22/public/hw5/Makefile .
   ```

3. Copy the provided files from the *eecs22* account.

   ```
   cp ~eecs22/public/hw5/Image.c .
   cp ~eecs22/public/hw5/Image.h .
   cp ~eecs22/public/hw5/MovieLab.c .
   cp ~eecs22/public/hw5/Constants.h .
   cp ~eecs22/public/hw5/watermark.ppm .
   ```

   Here,

   - **Image.h** is a header file for the definition of the *IMAGE* structure and declarations of the pixel mapping functions we have been using for Assignment 4. We also added the *YUVIMAGE* structure and the corresponding pixel mapping functions for the **YUV** color space;
   - **Image.c** is a modified source code file for *Image.h*;
   - **MovieLab.c** is a template file with sample code for command-line argument parsing, and the basic file I/O functions.

- **Constants.h** is a modified header file which contains macros used in this assignment.
- **watermark.ppm** is a watermark image used in this assignment.

All the files listed above will be available in ~`eecs22/public/hw5/` after the deadline of Assignment 4.

4. Create a symbolic link to the input movie stream file from the *eecs22* account on the *zuma* or *crystalcove* server.

```
ln -s ~eecs22/public/hw5/dive.yuv
```

Here, **dive.yuv** is a symbolic link to the input movie file in our *eecs22* account. Since we have space limitation for each account on the servers, it is helpful to save disk space for each account by sharing the read-only input file.

We will use the *dive.yuv* file as the test input stream for this assignment. Once a movie operation is done, you can save the output movie as *name.yuv* in your working directory by using the *"-o"* option.

You will need a *YUV* player to view the movie files. Our provided *YUV* player requires you to have X window support on your own machine where you use either **PuTTY** (Windows users) or **Terminal** (Mac Users) to remote login to the Linux server. For Mac users, you need to have **XQuartz** installed to support the X window. Please remember to add the *"-X"* option while using the *"ssh"* command (then macOS will ask you to install **XQuartz** if it is missing). For Windows users, you need to install the X server first and set the configurations in **PuTTY** with proper *X11 forwarding*. A free X server, **Xming**, for the Windows system is available from `https://sourceforge.net/projects/xming/`. The detailed instructions on **PuTTY** configuration is available from `http://www.geo.mtu.edu/geoschem/docs/putty_install.html`.

With the X server running properly, you can use the following commands to play your movie files (.yuv):

```
cd hw5
~eecs22/bin/yay -s WIDTHxHEIGHT filename.yuv
```

Specifically, you can play the test video stream by using:

```
~eecs22/bin/yay -s 320x240 dive.yuv
```

1.3 Design the MovieLab Program

In this assignment, we will design a data structure to represent the movie in a C program. Fig. 1 illustrates the double linked list data structure for the movie in this assignment.

1.3.1 The Image.c Module (Provided)

In Assignment 4, we designed the *Image.c* module for the basic image manipulation functions. A struct *IMAGE* is defined for the pixels in the **RGB** format. Also, image creation/deletion functions and pixel intensity Get/Set functions are defined accordingly.

Since the data structure for the **YUV** format is almost the same as the **RGB** format, we define similar structure and image manipulation functions for the **YUV** format. Now the structures and function signatures in *Image.h* look like this:

```
typedef struct {
    unsigned int Width;   /* Image width */
    unsigned int Height;  /* Image height */
    unsigned char *R;     /* Pointer to the memory storing */
                          /* all the R intensity values */
    unsigned char *G;     /* Pointer to the memory storing */
                          /* all the G intensity values */
```

```
    unsigned char *B;     /* Pointer to the memory storing */
                          /* all the B intensity values */
} IMAGE;

/* Get the intensity value of the Red channel of pixel (x, y) */
/* in the RGB image */
unsigned char GetPixelR(const IMAGE *image, unsigned int x, unsigned int y);

/* Get the intensity value of the Green channel of pixel (x, y) */
/* in the RGB image */
unsigned char GetPixelG(const IMAGE *image, unsigned int x, unsigned int y);

/* Get the intensity value of the Blue channel of pixel (x, y) */
/* in the RGB image */
unsigned char GetPixelB(const IMAGE *image, unsigned int x, unsigned int y);

/* Set the intensity value of the Red channel of pixel (x, y) */
/* in the RGB image with valueR */
void SetPixelR(IMAGE *image, unsigned int x, unsigned int y,
               unsigned char valueR);

/* Set the intensity value of the Green channel of pixel (x, y) */
/* in the RGB image with valueG */
void SetPixelG(IMAGE *image, unsigned int x, unsigned int y,
               unsigned char valueG);

/* Set the intensity value of the Blue channel of pixel (x, y) */
/* in the RGB image with valueB */
void SetPixelB(IMAGE *image, unsigned int x, unsigned int y,
               unsigned char valueB);

/* Allocate the memory space for the RGB image and the memory spaces */
/* for the RGB intensity values. Return the pointer to the RGB image. */
IMAGE *CreateImage(unsigned int width, unsigned int height);

/* Release the memory spaces for the RGB intensity values. */
/* Release the memory space for the RGB image. */
void DeleteImage(IMAGE *image);

typedef struct {
    unsigned int Width;   /* Image width */
    unsigned int Height;  /* Image height */
    unsigned char *Y;     /* Pointer to the memory storing */
                          /* all the Y intensity values */
    unsigned char *U;     /* Pointer to the memory storing */
                          /* all the U intensity values */
    unsigned char *V;     /* Pointer to the memory storing */
                          /* all the V intensity values */
} YUVIMAGE;

/* Get the intensity value of the Y channel of pixel (x, y) */
/* in the YUV image */
unsigned char GetPixelY(const YUVIMAGE *YUVimage, unsigned int x, unsigned int y);
```

```
/* Get the intensity value of the U channel of pixel (x, y) */
/* in the YUV image */
unsigned char GetPixelU(const YUVIMAGE *YUVimage, unsigned int x, unsigned int y);

/* Get the intensity value of the V channel of pixel (x, y) */
/* in the YUV image */
unsigned char GetPixelV(const YUVIMAGE *YUVimage, unsigned int x, unsigned int y);

/* Set the intensity value of the Y channel of pixel (x, y) */
/* in the YUV image with valueY */
void SetPixelY(YUVIMAGE *YUVimage, unsigned int x, unsigned int y,
               unsigned char valueY);

/* Set the intensity value of the U channel of pixel (x, y) */
/* in the YUV image with valueU */
void SetPixelU(YUVIMAGE *YUVimage, unsigned int x, unsigned int y,
               unsigned char valueU);

/* Set the intensity value of the V channel of pixel (x, y) */
/* in the YUV image with valueV */
void SetPixelV(YUVIMAGE *YUVimage, unsigned int x, unsigned int y,
               unsigned char valueV);

/* Allocate the memory space for the YUV image and the memory spaces */
/* for the YUV intensity values. Return the pointer to the YUV image. */
YUVIMAGE *CreateYUVImage(unsigned int width, unsigned int height);

/* Release the memory spaces for the YUV intensity values. */
/* Release the memory space for the YUV image. */
void DeleteYUVImage(YUVIMAGE *YUVimage);
```

1.3.2 The ImageList.c Module

Next we are going to design a double-linked list to store the frames (images) for the movie and keep them in the correct order.

As discussed in **Lecture 19** and **20**, a double-linked list is a data structure that consists of a set of sequentially linked records called *entries*. Each *entry* contains at least two fields, called links, that are references to the previous (*Prev*) and to the next (*Next*) entry in the sequence of entries. The first (*First*) and last (*Last*) entries' *Prev* and *Next* links, respectively, point to a terminator, *NULL*, to facilitate easy traversal of the list.

Please add one module **ImageList.c** (with a header file **ImageList.h**) to your *MovieLab* program.

In this module, define the following two structures:

- The structure for the image list entry **IENTRY**:

```
typedef struct ImageEntry IENTRY;
typedef struct ImageList ILIST;

struct ImageEntry {
    ILIST *List;        /* Pointer to the list which this entry belongs to */
    IENTRY *Next;       /* Pointer to the next entry, or NULL */
    IENTRY *Prev;       /* Pointer to the previous entry, or NULL */
```

```
    IMAGE *RGBImage;     /* Pointer to the RGB image, or NULL */
    YUVIMAGE *YUVImage;  /* Pointer to the YUV image, or NULL */
};
```

Note that either the *RGBImage* pointer or the *YUVImage* pointer will be *NULL* at any time. The *YUVImage* pointer will be valid (and *RGBImage* is *NULL*) when loading and saving the movie file, and the *RGBImage* pointer is in use (so *YUVImage* is *NULL*) when DIP operations take place. Please make sure that you free the memory space pointed to by the unused pointer.

- The structure for the image list **ILIST**:

```
struct ImageList {
    unsigned int Length;  /* Length of the list */
    IENTRY *First;        /* Pointer to the first entry, or NULL */
    IENTRY *Last;         /* Pointer to the last entry, or NULL */
};
```

In the same module, define the following double-linked list functions:

```
/* Create a new image list */
ILIST *CreateImageList(void);

/* Delete an image list (and all entries) */
void DeleteImageList(ILIST *list);

/* Insert a RGB image to the image list at the end */
void AppendRGBImage(ILIST *list, IMAGE *RGBimage);

/* Insert a YUV image to the image list at the end */
void AppendYUVImage(ILIST *list, YUVIMAGE *YUVimage);

/* Crop an image list */
void CropImageList(ILIST *list, unsigned int start, unsigned int end);

/* Fast forward an image list */
void FastImageList(ILIST *list, unsigned int factor);

/* Reverse an image list */
void ReverseImageList(ILIST *list);
```

Note: Please refer to the slides of **Lecture 19** and **20** for an example of implementing a double-linked list.

1.3.3 The Movie.c Module

Please add one module **Movie.c** (with a header file **Movie.h**) to handle basic operations on the movie.

- The **MOVIE** struct: We will use a *struct* type to aggregate all the information of one movie. Please define the following struct in **Movie.h**:

```
/* the movie structure */
typedef struct {
    ILIST *Frames; /* Pointer to the frame list */
} MOVIE;
```

- Define the following functions for basic movie operations. Please use the following function prototypes (in **Movie.h**) and define the functions properly (in **Movie.c**)

```
/* Allocate the memory space for the movie and the memory space */
/* for the frame list. Return the pointer to the movie. */
MOVIE *CreateMovie(void);

/* Release the memory space for the frame list. */
/* Release the memory space for the movie. */
void DeleteMovie(MOVIE *movie);

/* Convert a YUV movie to a RGB movie */
void YUV2RGBMovie(MOVIE *movie);

/* Convert a RGB movie to a YUV movie */
void RGB2YUVMovie(MOVIE *movie);
```

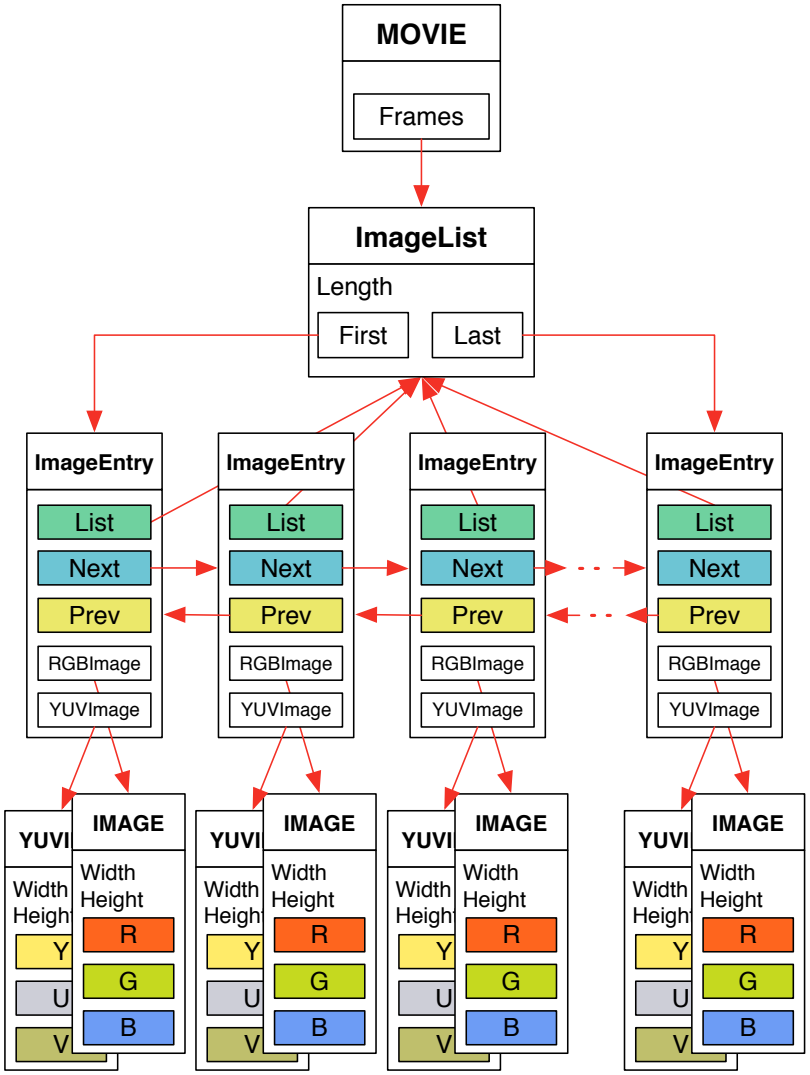

Figure 1: Double Linked List for the Movie

- Conversion between YUV and RGB:
  The conversion between the YUV format (used by many image and movie compression methods) and the RGB format (used by many hardware manufacturers) can be done by the following formulas. They show how to compute a pixel's values in one format from the pixel values in the other format.

  Please use the following formulas for the *YUV2RGBMovie* and *RGB2YUVMovie* functions.

  – Conversion from RGB to YUV:

```
Y = clip( ( (  66 * R + 129 * G +  25 * B + 128) >> 8) +  16 )
U = clip( ( ( -38 * R -  74 * G + 112 * B + 128) >> 8) + 128 )
V = clip( ( ( 112 * R -  94 * G -  18 * B + 128) >> 8) + 128 )
```

  – Conversion from YUV to RGB:

```
C = Y - 16
D = U - 128
E = V - 128
R = clip(( 298 * C           + 409 * E + 128) >> 8)
G = clip(( 298 * C - 100 * D - 208 * E + 128) >> 8)
B = clip(( 298 * C + 516 * D           + 128) >> 8)
```

  Here, $clip()$ denotes clipping a value to the range of 0 to 255 (saturated arithmetic). More specifically,

```
clip(x) = x, if 0 <= x <= 255;
clip(x) = 0, if x < 0;
clip(x) = 255, if x > 255.
```

  NOTE: Use type *int* for the variables in the calculation.

1.3.4 The MovieLab.c Module

Extend the **MovieLab.c** template as the top module of the *MovieLab* program.

- Support for command-line arguments:
  The C language provides a method to pass arguments to the main() function when executing the program. This is typically accomplished by specifying arguments on the operating system command line (console).

  Here, the prototype for *main()* looks like:

```
int main(int argc, char *argv[])
{
    ...
}
```

  There are two parameters in the *main()* function. The first parameter (*int argc*) is the number of items on the command line, including the executable name and all the arguments. Each argument on the command line is separated by one or more spaces, and the operating system places each argument directly into its own null-terminated string. The second parameter (*char *argv[]*) of *main()* is an array of pointers to the character strings containing each argument.

  Please add support for command-line arguments to the *MovieLab.c* program. The following options should be supported:

  – **-i** <**file**> to provide the input <file> name
  – **-o** <**file**> to provide the output <file> name
  – **-f** <**framenum**> to determine how many frames are read from the input stream

- **-s** <**WIDTHxHEIGHT**> to set the resolution of the input stream (WIDTHxHEIGHT)
- **-aging** to activate the aging filter
- **-hflip** to activate horizontal flip
- **-edge** to activate the edge filter
- **-crop** <**start-end**> to crop the movie frames from <start> to <end>
- **-fast** <**factor**> to fast forward the movie by <factor> (1+)
- **-rvs** to reverse the frame order of the input movie
- **-watermark** <**file**> to add a watermark from <file> to every movie frame
- **-spotlight** <**radius**> to spotlight a circle of <radius> on every movie frame
- **-zoom** (BONUS) to zoom in and out the input movie
- **-h** to display this usage information

The *MovieLab.c* template file contains the sample code for the support of **"-i"**, **"-o"** and **"-h"** options. Please extend the code accordingly to support the rest of the options.

NOTE: The *MovieLab* program can perform multiple operations in an execution. If the user gives more than one option, please perform the selected options in the following order: **"-aging"**, **"-hflip"**, **"-edge"**, **"-crop"**, **"-fast"**, **"-rvs"**, **"-watermark"**, **"-spotlight"**, and then **"-zoom"**.

The **"-i"**, **"-o"**, **"-f"**, **"-s"** options are mandatory to *MovieLab* with an exception when the user just wants to see the usage information (option **"-h"**).

Please show proper warning messages and terminate the execution of *MovieLab* if any of the mandatory options are missing as the command-line argument.

In order to get two integer values for the **"-s"** option, please use the following piece of code (assume that the *i*th command-line argument contains these two values):

```
unsigned int width, height;
if (sscanf(argv[i], "%ux%u", &width, &height) == 2) {
    /* input is correct                    */
    /* the image width is stored in width  */
    /* the image height is stored in height */
} else {
    /* input format error */
}
```

You can search online for the synopsis and description of the *sscanf()* function. Basically, this function reads formatted data from a character string and returns the number of items in the argument list successfully filled.

If we run the *MovieLab* with the **"-h"** option, we will have:

```
% ./MovieLab -h

Usage: MovieLab -i <file> -o <file> -f <framenum> -s <WIDTHxHEIGHT> [options]
Options:
-aging                Activate the aging filter on every movie frame
-hflip                Activate horizontal flip on every movie frame
-edge                 Activate the edge filter on every movie frame
-crop <start-end>     Crop the movie frames from <start> to <end>
-fast <factor>        Fast forward the movie by <factor>
-rvs                  Reverse the frame order of the input movie
-watermark <file>     Add a watermark from <file> to every movie frame
-spotlight <radius>   Spotlight a circle of <radius> on every movie frame
-zoom                 Zoom in and out the input movie
-h                    Display this usage information
```

Otherwise, we need to run the *MovieLab* with proper information for the movie and operation options, e.g:

```
% ./MovieLab -i dive -o out -f 180 -s 320x240 -aging
The movie file dive.yuv has been read successfully!
Operation Aging is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

- Load and save movie files:
  We have provided some file I/O functions defined in the *MovieLab.c* module. The function signatures for the file I/O functions are:
  - **YUVIMAGE* LoadOneFrame(const char* fname, int n, unsigned int width, unsigned height)**
    loads the movie file with name *fname.yuv*, and returns the pointer to a *YUVIMAGE* struct which contains the color intensities for channel Y, U and V of the *n*-th frame.
  - **int SaveMovie(const char *fname, MOVIE *movie)**
    opens the movie file with name *fname.yuv*, and saves the movie frames into *fname.yuv*.
  - **MOVIE *LoadMovie(const char *fname, int frameNum, unsigned int width, unsigned height)**
    loads a number *frameNum* of frames from the movie file with name *fname.yuv*, and returns the pointer to the *movie* struct. **NOTE: You need to implement this function. Please call the *LoadOneFrame()* and *AppendYUVImage()* functions to implement the function**.

  Inside *LoadMovie()* which gets the content of the input movie file, you need to first allocate the memory space for the *movie* struct by calling *CreateMovie()*. The *LoadOneFrame()* function will take the file name of the video, the resolution of the video, and the frame index to be read as pass-in arguments, and return a *YUVIMAGE* pointer to the memory space storing the input frame. At the end of your program, you need to free these memory spaces to avoid memory leakage.

- Perform DIP operations on the movie:
  We will add support for 8 DIP operations on the movie file:
  - Create an aging movie (the **"-aging"** option):
    Traverse the frame list of the movie, and perform *Aging()* operation on each frame image. The execution of our program should be like:

    ```
    % ./MovieLab -i dive -o out -s 320x240 -f 180 -aging
    The movie file dive.yuv has been read successfully!
    Operation Aging is done!
    The movie file out.yuv has been written successfully!
    180 frames are written to the file out.yuv in total.
    ```

– Flip the movie horizontally (the **"-hflip"** option):
Traverse the frame list of the movie, and perform *HFlip()* operation on each frame image. The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -hflip
The movie file dive.yuv has been read successfully!
Operation HFlip is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

– Create a edge-detected movie (the **"-edge"** option):
Traverse the frame list of the movie, and perform *Edge()* operation on each frame image. The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -edge
The movie file dive.yuv has been read successfully!
Operation Edge is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

– Crop frames from the movie (the **"-crop"** option):
Perform the *CropImageList()* operation on the *ImageList* in the *movie* structure. Your program should print the number of frames after cropping. Fig. 2 illustrates the concept of cropping operation. In this example, the program takes frame 71 to frame 140 and generates a new movie with 70 frames.

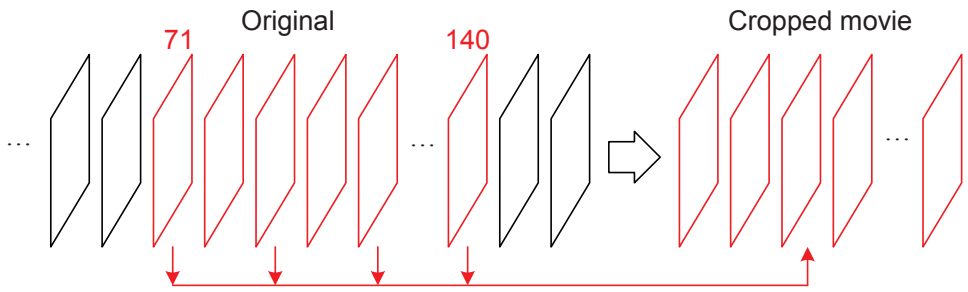

Figure 2: Cropping Operation

NOTE: The frame index in the movie starts from 0, and the two frames indexed by *start* and *end* are kept in the new movie.

The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -crop 71-140
The movie file dive.yuv has been read successfully!
Operation Crop is done! New number of frames is 70.
The movie file out.yuv has been written successfully!
70 frames are written to the file out.yuv in total.
```

– Create a fast forwarded movie (the **"-fast"** option):
Perform the *FastImageList()* operation on the *ImageList* in the *movie* structure with the given fast forward factor. Note that your program should also print the number of frames after fast forwarding. Fig. 3 illustrates the concept of fast forward operation. In this example of fast forwarding by 3, the program will take every third frame from the original one to generate the new movie.

The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -fast 3
The movie file dive.yuv has been read successfully!
```

```
Operation Fast Forward is done! New number of frames is 60.
The movie file out.yuv has been written successfully!
60 frames are written to the file out.yuv in total.
```

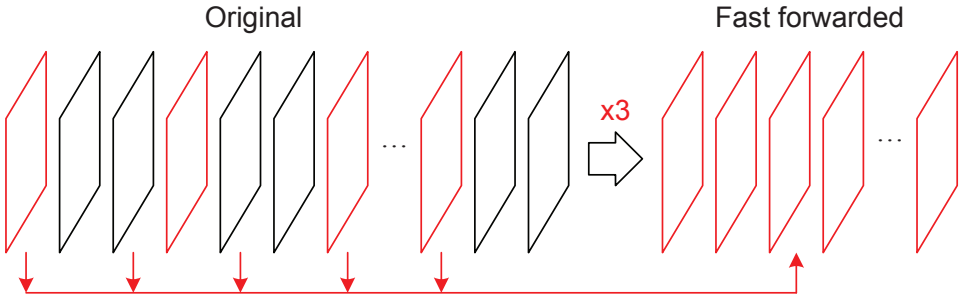

Figure 3: Fast Forwarding Operation

– Reverse the frame order in the movie (the **"-rvs"** option):
Perform the *ReverseImageList()* operation on the *ImageList* in the *movie* structure. The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -rvs
The movie file dive.yuv has been read successfully!
Operation Reverse is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

– Add a watermark to each frame in the movie (the **"-watermark"** option):
Traverse the frame list of the movie, and add a watermark to each frame image. The position of the watermark will be decided at run time by using the random number generator, and it stays at the same place for *15* consecutive frames. Afterwards, it will move to a new position. Note that the watermark can be at any position in the frame image. You should use the random number generator to generate the coordinates of the top left corner of the watermark image in the frame image. The coordinates can be any valid coordinates in the frame image. Just ignore any part of the watermark image which is out of the boundary of the frame image.

NOTE: The loaded watermark image can be of any size. So this operation is a little different from the *Watermark()* function in Assignment 3. Please check the function prototype of the new *Watermark()* function in Section 4.2.

The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -watermark watermark
The movie file dive.yuv has been read successfully!
Operation Watermark is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

– Spotlight a circle on each frame in the movie (the **"-spotlight"** option):
Traverse the frame list of the movie, and spotlight a circle on each frame image. Here, for any pixel that is within the circle, i.e. the distance from the pixel to the center is less than or equal to the radius provided by the user, your program keeps its original intensity values. For all pixels that are out of the circle, your program makes them black. The center of the circle starts from $(0,0)$, and moves as indicated in Fig. 4. The movement speed is 10 pixels per frame, i.e. $(0,0),(10,10),(20,20),\ldots$. When the center reaches any boundary of the frame image, it will bounce to another direction as in Fig. 4. For example, if the image size is 50×40, then the positions of the center will be: $(0,0)$, $(10,10)$, $(20,20)$, $(30,30)$, $(40,40)$, $(50,30)$,

$(40, 20)$, $(30, 10)$, ... Note that the radius can be any nonnegative integer.

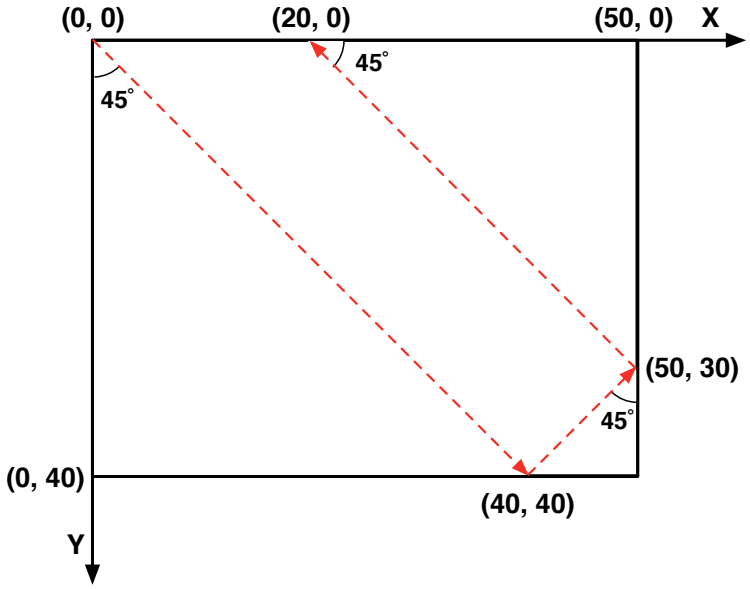

Figure 4: Spotlight Movement

The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -spotlight 100
The movie file dive.yuv has been read successfully!
Operation Spotlight is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

HINT: We will reuse some functions defined in *DIPs.c* and *Advanced.c* from Assignment 4. Please integrate these two modules into a new *DIPs.c* module for this assignment. You also have to adjust your Makefile accordingly with proper targets and dependencies and include the header file (*DIPs.h*) properly in your source code.

NOTE: Due to the space limitation for the account on the Linux server, please always use the same output file name, i.e. out.yuv, when you test your program so as to save disk space.

For references, we put the output movie files from the above 8 operations (plus the *Zoom* operation from Section 2) in the shared folder. You may compare your results with them visually.

```
~eecs22/public/hw5/demo/aging.yuv
~eecs22/public/hw5/demo/hflip.yuv
~eecs22/public/hw5/demo/edge.yuv
~eecs22/public/hw5/demo/crop.yuv
~eecs22/public/hw5/demo/fast.yuv
~eecs22/public/hw5/demo/rvs.yuv
~eecs22/public/hw5/demo/watermark.yuv
~eecs22/public/hw5/demo/spotlight.yuv
~eecs22/public/hw5/demo/zoom.yuv
```

2 Bonus (10 points)

Extend the **MovieLab** program with an additional DIP operation (the **"-zoom"** option) on the movie.

In the *Zoom* operation, your program traverses the frame list of the movie, and resizes each frame image and puts the result in the center of the frame. For any remaining pixels in the frame, make them black. At the beginning of the movie, the resize percentage is 0%, and then it increases by 2% per frame. When the resize percentage reaches 100%, it then decreases by 2% per frame until 0. Afterwards, the percentage increases again. So a sequence of the resize percentages will be: $0\%, 2\%, 4\%, 6\%, ..., 98\%, 100\%, 98\%, 96\%, ..., 4\%, 2\%, 0\%, 2\%, 4\%, ...$ You can assume that the resize percentage will always be an even number.

The execution of our program should be like:

```
% ./MovieLab -i dive -o out -s 320x240 -f 180 -zoom
The movie file dive.yuv has been read successfully!
Operation Zoom is done!
The movie file out.yuv has been written successfully!
180 frames are written to the file out.yuv in total.
```

HINT: This operation is actually a combination of *Resize()* and *AddBorder()*. Perform these two operations on each frame image.

3 Build the Makefile

Please create your own **Makefile** with at least the following targets:

- **all:** the dummy target to generate the executable program *MovieLab*.
- **clean:** the target to clean all the intermediate files, e.g. object files, the output .yuv file, and the executable program.
- ***.o:** the target to generate the object file *.o from the C source code file *.c.
- **MovieLab:** the target to generate the executable program *MovieLab*.

4 Implementation Details

Here is a recap of all the structures and functions you need to implement in this assignment.

4.1 Structure Definitions

For this assignment, you need to define the following structures in **ImageList.h**:

```
typedef struct ImageEntry IENTRY;
typedef struct ImageList ILIST;

struct ImageEntry {
    ILIST *List;         /* Pointer to the list which this entry belongs to */
    IENTRY *Next;        /* Pointer to the next entry, or NULL */
    IENTRY *Prev;        /* Pointer to the previous entry, or NULL */
    IMAGE *RGBImage;     /* Pointer to the RGB image, or NULL */
    YUVIMAGE *YUVImage;  /* Pointer to the YUV image, or NULL */
};

struct ImageList {
```

```
    unsigned int Length;  /* Length of the list */
    IENTRY *First;        /* Pointer to the first entry, or NULL */
    IENTRY *Last;         /* Pointer to the last entry, or NULL */
};
```

The following structure in **Movie.h**:

```
/* the movie structure */
typedef struct {
    ILIST *Frames; /* Pointer to the frame list */
} MOVIE;
```

4.2 Function Prototypes

For this assignment, you need to define the following functions in the **ImageList.c** module:

```
/* Create a new image list */
ILIST *CreateImageList(void);

/* Delete an image list (and all entries) */
void DeleteImageList(ILIST *list);

/* Insert a RGB image to the image list at the end */
void AppendRGBImage(ILIST *list, IMAGE *RGBimage);

/* Insert a YUV image to the image list at the end */
void AppendYUVImage(ILIST *list, YUVIMAGE *YUVimage);

/* Crop an image list */
void CropImageList(ILIST *list, unsigned int start, unsigned int end);

/* Fast forward an image list */
void FastImageList(ILIST *list, unsigned int factor);

/* Reverse an image list */
void ReverseImageList(ILIST *list);
```

The following functions in the **Movie.c** module:

```
/* Allocate the memory space for the movie and the memory space */
/* for the frame list. Return the pointer to the movie. */
MOVIE *CreateMovie(void);

/* Release the memory space for the frame list. */
/* Release the memory space for the movie. */
void DeleteMovie(MOVIE *movie);

/* Convert a YUV movie to a RGB movie */
void YUV2RGBMovie(MOVIE *movie);

/* Convert a RGB movie to a YUV movie */
void RGB2YUVMovie(MOVIE *movie);
```

The following functions in the **MovieLab.c** module:

```
/* Load the movie frames from the input file */
MOVIE *LoadMovie(const char *fname, int frameNum,
                 unsigned int width, unsigned height);

/* Main function */
int main(int argc, char *argv[]);
```

The DIP functions in the **DIPs.c** module:

```
/* Aging */
IMAGE *Aging(IMAGE *image);

/* Horizontal flip */
IMAGE *HFlip(IMAGE *image);

/* Edge detection */
IMAGE *Edge(IMAGE *image);

/* Add a watermark to an image */
IMAGE *Watermark(IMAGE *image, const IMAGE *watermark,
                 unsigned int topLeftX, unsigned int topLeftY);

/* Spotlight */
IMAGE *Spotlight(IMAGE *image, int centerX, int centerY, unsigned int radius);

/* Zoom an image */
IMAGE *Zoom(IMAGE *image, unsigned int percentage);
```

You can reuse some functions (*Aging()*, *HFlip()* and *Edge()*) from the previous assignments. Then you need to define other functions in **DIPs.c** as well.

5 Budgeting Your Time

You have two weeks to complete this assignment, but we encourage you to get started early. We suggest you budget your time as follows:

- Week 1:
  1. Build **Makefile**.
  2. Complete *CreateImageList*, *DeleteImageList*, *AppendRGBImage* and *AppendYUVImage* in **ImageList.c** and **ImageList.h**.
  3. Complete *CreateMovie*, *DeleteMovie*, *YUV2RGBMovie* and *RGB2YUVMovie* in **Movie.c** and **Movie.h**.
  4. Complete *LoadMovie* in **MovieLab.c**.
  5. Copy previous DIP functions (*Aging*, *HFlip* and *Edge*) to **DIPs.c** and **DIPs.h**, and adjust them if necessary.
  6. Add the command-line argument support in the *main* function.

  Now you can compile and run your *MovieLab* to test the basic DIP operations, provided that you have empty function definitions for all undefined functions. Also, run your program in *Valgrind* to detect any memory leaks and invalid memory accesses. Fix any problem reported by *Valgrind*.

- Week 2:
    1. Complete *ReverseImageList*, *FastImageList* and *CropImageList* in **ImageList.c** and **ImageList.h**.
    2. Complete the remaining DIP functions (*Watermark*, *Spotlight* and *Zoom*) in **DIPs.c** and **DIPs.h**.
    3. Use *Valgrind* to check memory usage. Fix the code if *Valgrind* complains about any errors or memory leaks.
    4. Script the result of your program and submit your work.

6 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 *MovieLab* by using your **Makefile**
3. Run the program: *% MovieLab -h*
4. Run the program: *% MovieLab -i dive -o out -f 50 -s 320x240 -aging -hflip*
5. Run the program: *% MovieLab -i dive -o out -f 100 -s 320x240 -edge*
6. Run the program: *% MovieLab -i dive -o out -f 180 -s 320x240 -fast 3 -crop 101-160*
7. Run the program: *% MovieLab -i dive -o out -f 100 -s 320x240 -rvs*
8. Run the program: *% MovieLab -i dive -o out -f 150 -s 320x240 -watermark watermark* under the monitor of *Valgrind*
9. Run the program: *% MovieLab -i dive -o out -f 150 -s 320x240 -spotlight 100* under the monitor of *Valgrind*
10. (Optional) Run the program: *% MovieLab -i dive -o out -f 100 -s 320x240 -zoom*
11. Clean all the object files, output .yuv file and executable program by using your **Makefile**.
12. Stop the script by typing the command: *exit*.
13. Rename the script file to *MovieLab.script*.

NOTE: The script file is important, and will be checked in grading. You must follow the above steps to create the script file. ***Again, please do not open any text editor while scripting!!!***

7 Submission

Go to the parent directory of your *hw5* folder, and turn in your homework by running:

```
% ~eecs22/bin/turnin.sh
```

Your *hw5* folder should contain the following files as the whole package of your program:

- *MovieLab.c*
- *MovieLab.script*
- *MovieLab.txt*
- *Movie.c*
- *Movie.h*

- *ImageList.c*
- *ImageList.h*
- *Image.c*
- *Image.h*
- *DIPs.c*
- *DIPs.h*
- *FileIO.c*
- *FileIO.h*
- *Constants.h*
- *Makefile*

8 Grading

- Makefile (compilable, no warnings and errors): 10 points
- Support for command-line arguments: 15 points
- Struct ImageEntry, ImageList and related functions: 20 points
- Struct MOVIE and related functions: 10 points
- Watermark operation: 15 points
- Spotlight operation: 15 points
- All other operations and no valgrind errors: 15 points
- Bonus (the Zoom operation): 10 points