Constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

/* Global macros */
/* Maximum length of file names */
#define SLEN 80

#define MAX_PIXEL 255   /* max pixel value */
#define MIN_PIXEL 0     /* min pixel value */

#define PI 3.14159265358979323846264338327950288

#endif
```

DIPs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "DIPs.h"
#include "Image.h"
#include "Constants.h"
#include "FileIO.h"
#include <math.h>

/* Black and White function */
IMAGE *BlackNWhite(IMAGE *image, int percent)
{
```

```
assert(image);
int x, y, gray;
float p = percent / 100.0;
int offsetR, offsetG, offsetB;
int newR, newG, newB;

for (x = 0; x < image->W; x++)
{
   for(y = 0; y < image->H; y++)
   {
        /* get grayscale color */
      gray = (GetPixelR(image, x, y) + GetPixelG(image, x, y) + GetPixelB(image, x, y)) / 3;

        /* find difference from grayscale */
        offsetR = gray - GetPixelR(image, x, y);
        offsetG = gray - GetPixelG(image, x, y);
        offsetB = gray - GetPixelB(image, x, y);

        /* add difference multiplied by percentage */
        newR = GetPixelR(image, x, y) + (offsetR * p);
        newG = GetPixelG(image, x, y) + (offsetG * p);
        newB = GetPixelB(image, x, y) + (offsetB * p);

      newR = ((newR < 0) ? 0 : ((newR > 255) ? 255 : newR));
        newG = ((newG < 0) ? 0 : ((newG > 255) ? 255 : newG));
        newB = ((newB < 0) ? 0 : ((newB > 255) ? 255 : newB));

        /* Sets new color change to image */
```

```
            SetPixelR(image, x, y, newR);
            SetPixelG(image, x, y, newG);
            SetPixelB(image, x, y, newB);
        }
    }

    return image;
}

/* Hue function */
IMAGE *HueRotate(IMAGE *image, int percent)
{
    double degree = (percent / 100.0) * 360.0;
    double a, b, r;
    double d = degree * PI / 180.0;
    double tmpr, tmpg, tmpb;

    /* alpha, beta, rho equations */
    a = (2 * cos(d) + 1.0) / 3.0;
    b = (1.0 - cos(d)) / 3.0 - sin(d) / sqrt(3.0);
    r = (1.0 - cos(d)) / 3.0 + sin(d) / sqrt(3.0);

    for (int x = 0; x < image->W; x++)
    {
            for (int y = 0; y < image->H; y++)
        {
            tmpr = GetPixelR(image, x, y) * a + GetPixelG(image, x, y) * b + GetPixelB(image, x, y)
* r;
```

```
        tmpg = GetPixelR(image, x, y) * r + GetPixelG(image, x, y) * a + GetPixelB(image, x, y)
* b;
        tmpb = GetPixelR(image, x, y) * b + GetPixelG(image, x, y) * r + GetPixelB(image, x, y)
* a;

        SetPixelR(image, x, y, (tmpr > MAX_PIXEL)?MAX_PIXEL:(tmpr < 0)?0:tmpr);
        SetPixelG(image, x, y, (tmpg > MAX_PIXEL)?MAX_PIXEL:(tmpg < 0)?0:tmpg);
        SetPixelB(image, x, y, (tmpb > MAX_PIXEL)?MAX_PIXEL:(tmpb < 0)?0:tmpb);
      }
   }
   return image;
}
```

DIPs.h

```
#ifndef DIPS_H
#define DIPS_H

#include "Image.h"
#include "FileIO.h"

// BlackNWhite filter
IMAGE *BlackNWhite(IMAGE *image, int percent);

// HueRotate filter
IMAGE *HueRotate(IMAGE *image, int percent);

#endif
```

FileIO.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

#include "Constants.h"
#include "FileIO.h"
#include "Image.h"

IMAGE *LoadImage(const char *fname)
{
	FILE		*File;
	char		Type[SLEN];
	int		W, H, MaxValue;
	unsigned int x, y;
	char		fname_tmp[SLEN];
	IMAGE		*image;
	strcpy(fname_tmp, fname);
	File = fopen(fname_tmp, "r");
	if (!File) {
#ifdef DEBUG
		printf("\nCan't open file \"%s\" for reading!\n", fname);
#endif
		return NULL;
	}

	fscanf(File, "%79s", Type);
	if (Type[0] != 'P' || Type[1] != '6' || Type[2] != 0) {
```

```
#ifdef DEBUG
                printf("\nUnsupported file format!\n");
#endif
                fclose(File);
                return NULL;
        }

        fscanf(File, "%d", &W);
        if (W <= 0) {
#ifdef DEBUG
                printf("\nUnsupported image width %d!\n", W);
#endif
                fclose(File);
                return NULL;
        }

        fscanf(File, "%d", &H);
        if (H <= 0) {
#ifdef DEBUG
                printf("\nUnsupported image height %d!\n", H);
#endif
                fclose(File);
                return NULL;
        }

        fscanf(File, "%d", &MaxValue);
        if (MaxValue != 255) {
#ifdef DEBUG
```

```
                printf("\nUnsupported image maximum value %d!\n", MaxValue);
#endif
                fclose(File);
                return NULL;
        }
        if ('\n' != fgetc(File)) {
#ifdef DEBUG
                printf("\nCarriage return expected at the end of the file!\n");
#endif
                fclose(File);
                return NULL;
        }

        image = CreateImage(W, H);

        if (!image) {
#ifdef DEBUG
                printf("\nError creating image from %s!\n", fname_tmp);
#endif
                fclose(File);
                return NULL;
        }
        else {
                for (y = 0; y < image->H; y++)
                        for (x = 0; x < image->W; x++) {
                                SetPixelR(image, x, y, fgetc(File));
                                SetPixelG(image, x, y, fgetc(File));
                                SetPixelB(image, x, y, fgetc(File));
```

```
                }

            if (ferror(File)) {
#ifdef DEBUG
                printf("\nFile error while reading from file!\n");
#endif
                DeleteImage(image);
                return NULL;
            }

#ifdef DEBUG
            printf("%s was read successfully!\n", fname_tmp);
#endif
            fclose(File);
            return image;
        }
}
```

FileIO.h

```
#ifndef FILEIO_H
#define FILEIO_H

#include "Image.h"

/* Read an image from a file.                                   */
/* The size of the image needs to be pre-set.                   */
/* The memory space of the image will be allocated in this function. */
/* Return values:                                               */
```

```
/* NULL: fail to load or create an image                    */
/* image: load or create an image successfully              */
IMAGE *LoadImage(const char *fname);

#endif
```

Image.c

```
#include <stdlib.h>
#include <assert.h>
#include "Image.h"

/* 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)
{
        assert(image);

        assert(x < image->W);
        assert(y < image->H);

        assert(image->R);
        assert(image->G);
        assert(image->B);

        return image->R[x + y * image->W];
}

/* 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)
{
	assert(image);

	assert(x < image->W);
	assert(y < image->H);

	assert(image->R);
	assert(image->G);
	assert(image->B);

	return image->G[x + y * image->W];
}

/* 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)
{
	assert(image);

	assert(x < image->W);
	assert(y < image->H);

	assert(image->R);
	assert(image->G);
	assert(image->B);
```

```
	return image->B[x + y * image->W];
}

/* 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)
{
	assert(image);

	assert(x < image->W);
	assert(y < image->H);

	assert(image->R);
	assert(image->G);
	assert(image->B);

	image->R[x + y * image->W] = 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)
{
	assert(image);

	assert(x < image->W);
```

```
	assert(y < image->H);

	assert(image->R);
	assert(image->G);
	assert(image->B);

	image->G[x + y * image->W] = 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)
{
	assert(image);

	assert(x < image->W);
	assert(y < image->H);

	assert(image->R);
	assert(image->G);
	assert(image->B);

	image->B[x + y * image->W] = 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)
{
	IMAGE *image = (IMAGE *)malloc(sizeof(IMAGE));
	if (image == NULL) {
		return NULL;
	}

	image->W = width;
	image->H = height;

	image->R = (unsigned char*)malloc(width * height * sizeof(unsigned char));
	if (image->R == NULL) {
		free(image);
		return NULL;
	}

	image->G = (unsigned char*)malloc(width * height * sizeof(unsigned char));
	if (image->G == NULL) {
		free(image->R);
		free(image);
		return NULL;
	}

	image->B = (unsigned char*)malloc(width * height * sizeof(unsigned char));
	if (image->B == NULL) {
		free(image->G);
		free(image->R);
		free(image);
```

```
                return NULL;
        }

        return image;
}

/* Release the memory spaces for the RGB intensity values. */
/* Release the memory space for the RGB image. */
void DeleteImage(IMAGE *image)
{
        assert(image);
        assert(image->R);
        assert(image->G);
        assert(image->B);

        free(image->R);
        free(image->G);
        free(image->B);
        image->R = NULL;
        image->G = NULL;
        image->B = NULL;

        free(image);
}

/* 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)
```

```
{
	assert(YUVimage);

	assert(x < YUVimage->W);
	assert(y < YUVimage->H);

	assert(YUVimage->Y);
	assert(YUVimage->U);
	assert(YUVimage->V);

	return YUVimage->Y[x + y * YUVimage->W];
}

/* 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)
{
	assert(YUVimage);

	assert(x < YUVimage->W);
	assert(y < YUVimage->H);

	assert(YUVimage->Y);
	assert(YUVimage->U);
	assert(YUVimage->V);

	return YUVimage->U[x + y * YUVimage->W];
}
```

```
/* 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)
{
        assert(YUVimage);

        assert(x < YUVimage->W);
        assert(y < YUVimage->H);

        assert(YUVimage->Y);
        assert(YUVimage->U);
        assert(YUVimage->V);

        return YUVimage->V[x + y * YUVimage->W];
}

/* 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)
{
        assert(YUVimage);

        assert(x < YUVimage->W);
        assert(y < YUVimage->H);

        assert(YUVimage->Y);
```

```
	assert(YUVimage->U);
	assert(YUVimage->V);

	YUVimage->Y[x + y * YUVimage->W] = 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)
{
	assert(YUVimage);

	assert(x < YUVimage->W);
	assert(y < YUVimage->H);

	assert(YUVimage->Y);
	assert(YUVimage->U);
	assert(YUVimage->V);

	YUVimage->U[x + y * YUVimage->W] = 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)
{
```

```
	assert(YUVimage);

	assert(x < YUVimage->W);
	assert(y < YUVimage->H);

	assert(YUVimage->Y);
	assert(YUVimage->U);
	assert(YUVimage->V);

	YUVimage->V[x + y * YUVimage->W] = 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)
{
	YUVIMAGE *YUVimage = (YUVIMAGE *)malloc(sizeof(YUVIMAGE));
	if (YUVimage == NULL) {
		return NULL;
	}

	YUVimage->W = width;
	YUVimage->H = height;

	YUVimage->Y = (unsigned char*)malloc(width * height * sizeof(unsigned char));
	if (YUVimage->Y == NULL) {
		free(YUVimage);
		return NULL;
```

```
	}

	YUVimage->U = (unsigned char*)malloc(width * height * sizeof(unsigned char));
	if (YUVimage->U == NULL) {
		free(YUVimage->Y);
		free(YUVimage);
		return NULL;
	}

	YUVimage->V = (unsigned char*)malloc(width * height * sizeof(unsigned char));
	if (YUVimage->V == NULL) {
		free(YUVimage->U);
		free(YUVimage->Y);
		free(YUVimage);
		return NULL;
	}

	return YUVimage;
}

/* Release the memory spaces for the YUV intensity values. */
/* Release the memory space for the YUV image. */
void DeleteYUVImage(YUVIMAGE *YUVimage)
{
	assert(YUVimage);
	assert(YUVimage->Y);
	assert(YUVimage->U);
	assert(YUVimage->V);
```

```
	free(YUVimage->Y);
	free(YUVimage->U);
	free(YUVimage->V);
	YUVimage->Y = NULL;
	YUVimage->U = NULL;
	YUVimage->V = NULL;

	free(YUVimage);
}

IMAGE *CopyImage(const IMAGE *image)
{
	IMAGE *ret = CreateImage(image->W, image->H);

	for(unsigned int i = 0; i < image->W; i++)
	{
		for(unsigned int j = 0; j < image->H; j++)
		{
			SetPixelR(ret, i, j, GetPixelR(image, i, j));
			SetPixelG(ret, i, j, GetPixelG(image, i, j));
			SetPixelB(ret, i, j, GetPixelB(image, i, j));
		}
	}

	return ret;
}
```

Image.h

```
#ifndef IMAGE_H
#define IMAGE_H

typedef struct {
        unsigned int W;       /* Image width */
        unsigned int H;       /* 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;
```

ImageList.c

```
#include <stdlib.h>
#include <assert.h>
#include "ImageList.h"
#include "Image.h"

/* Create a new image entry */
IENTRY *CreateImageEntry(void)
{
    IENTRY *entry;
    entry = malloc(sizeof(IENTRY));

    if (!entry)
```

```
    {
        perror("Out of memory! Abort...");
        exit(10);
    }

    /* Sets all properties to NULL */
    entry->List = NULL;
    entry->Next = NULL;
    entry->Prev = NULL;
    entry->RGBImage = NULL;
    entry->YUVImage = NULL;

    return entry;
}

/* Delete image entry (and image objects)*/
void DeleteImageEntry(IENTRY *entry)
{
    assert(entry);

    if (entry->RGBImage)
    {
        DeleteImage(entry->RGBImage);
    }

    if (entry->YUVImage)
    {
        DeleteYUVImage(entry->YUVImage);
```

```
    }

    free(entry);
    entry = NULL;
}

/* Create a new image list */
ILIST *CreateImageList(void)
{
    ILIST *list;
    list = malloc(sizeof(ILIST));

    if (!list)
    {
        perror("Out of memory! Aborting...");
        exit(10);
    }

    list->length = 0;
    list->First = NULL;
    list->Last = NULL;

    return list;
}

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

```
    IENTRY *entry, *next;

    assert(list);

    entry = list->First;

    while (entry)
    {
          next = entry->Next;
       DeleteImageEntry(entry);
       entry = next;
       list->length--;
    }

    free(list);
    list = NULL;
}

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

    assert(list);
    assert(RGBimage);

    entry = CreateImageEntry();
    entry->List = list;
```

```
    if (list->Last)
    {
        entry->Next = NULL;
        entry->Prev = list->Last;
        list->Last->Next = entry;
        entry->RGBImage = RGBimage;
        list->Last = entry;
    }

    /* Empty List */
    else
    {
            entry->Next = NULL;
        entry->Prev = NULL;
        entry->RGBImage = RGBimage;
        list->First = entry;
            list->Last = entry;
    }

    list->length++;
}

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

```
    assert(list);
    assert(YUVimage);

    entry = CreateImageEntry();
    entry->List = list;

    if (list->Last)
    {
       entry->Next = NULL;
       entry->Prev = list->Last;
          list->Last->Next = entry;
       entry->YUVImage = YUVimage;
          list->Last = entry;
    }

    /* Empty List */
    else
    {
       entry->Next = NULL;
       entry->Prev = NULL;
       entry->YUVImage = YUVimage;
       list->First = entry;
          list->Last = entry;
    }

    list->length++;
}
```

```
/* Reverse an image list */
void ReverseImageList(ILIST *list)
{
    IENTRY *entry = NULL, *next = NULL, *prev = NULL;
    IENTRY *temp = NULL;
    entry = list->First;

    temp = list->First;
    list->First = list->Last;
    list->Last = temp;

    while (entry)
    {
        next = entry->Next;
       entry->Next = prev;
       prev = entry;
       entry->Prev = next;
       entry = next;
    }
}

/* Copy YUV Image */
YUVIMAGE *CopyYUVImage(YUVIMAGE *image)
{
      YUVIMAGE *ret = CreateYUVImage(image->W, image->H);

      for(unsigned int i = 0; i < image->W; i++)
      {
```

```
            for(unsigned int j = 0; j < image->H; j++)
            {
                SetPixelY(ret, i, j, GetPixelY(image, i, j));
                SetPixelU(ret, i, j, GetPixelU(image, i, j));
                SetPixelV(ret, i, j, GetPixelV(image, i, j));
            }
        }

        return ret;
}
```

ImageList.h

```
#ifndef IMAGELIST_H
#define IMAGELIST_H

#include <stdio.h>
#include "Image.h"

typedef struct ImageEntry IENTRY;
typedef struct ImageList ILIST;

struct ImageEntry {
        IMAGE *RGBImage;
        YUVIMAGE *YUVImage;
        IENTRY *Next;
        IENTRY *Prev;
        ILIST *List;
};
```

```
struct ImageList {
        IENTRY *First;
        IENTRY *Last;
        int length;
};

/* Create a new image entry */
IENTRY *CreateImageEntry(void);

/* Delete an image entry (and all contained images) */
void DeleteImageEntry(IENTRY *entry);

/* 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);

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

```
/* Copy YUV Image */
YUVIMAGE *CopyYUVImage(YUVIMAGE *image);

#endif
```

IterativeFilter.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include "IterativeFilter.h"
#include "Image.h"
#include "Movie.h"

MOVIE *doIterativeFilter(IMAGE *image, iterableFilter filter, int start, int end, int change)
{
   assert(image);
   MOVIE *movie = CreateMovie();
   IMAGE *RGBimage = CopyImage(image);

   int i = 0;

   /* Increasing Percentage */
   if (start < end)
   {
         change = ((change < 0) ? -change : change);
         for (i = start; i <= end; i += change)
         {
```

```
            AppendRGBImage(movie->Frames, filter(CopyImage(RGBimage), i));
        }
    }

    /* Decreasing Percentage */
    else
    {
        change = ((change > 0) ? -change : change);
        for (i = start; i >=end; i += change)
        {
            AppendRGBImage(movie->Frames, filter(CopyImage(RGBimage), i));
        }
    }

    DeleteImage(RGBimage);

    return movie;
}
```

IterativeFilter.h

```
#ifndef ITERATIVEFILTER_H
#define ITERATIVEFILTER_H

#include "Image.h"
#include "Movie.h"

// Typedef for iterableFilter
/* iterableFilter function pointer */
```

```
typedef IMAGE * (*iterableFilter)(IMAGE *image, int parameter);

// Function declaration for doIterativeFilter
/* Generate movie from input image by applying filter with parameter from <start> to <end>
using <step> variation */
MOVIE *doIterativeFilter(IMAGE *image, iterableFilter filter, int start, int end, int change);

#endif
```

Movie.c

```
#include <stdlib.h>
#include <assert.h>
#include "Movie.h"
#include "Image.h"
#include "ImageList.h"

/* Clip Function */
int clip(int x);

/* Allocate the memory space for the movie and the memory space */
/* for the frame list. Return the pointer to the movie. */
MOVIE *CreateMovie(void)
{
    MOVIE *movie;
    movie = malloc(sizeof(MOVIE));
    movie->Frames = CreateImageList();
    return movie;
}
```

```
/* Release the memory space for the frame list. */
/* Release the memory space for the movie. */
void DeleteMovie(MOVIE *movie)
{
    assert(movie);
    DeleteImageList(movie->Frames);
    free(movie);
    movie = NULL;
}

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

    IENTRY *entry;

    entry = movie->Frames->First;

    int x = 0;
    int i = 0, w = 0, h = 0;
    int c, d, e;

    /* Loops through the frames of the movie */
    for (i = 0; i < MovieLength(movie); i++)
    {
        entry->RGBImage = CreateImage(MovieWidth(movie), MovieHeight(movie));
```

```
        /* Seting new values to a new RGB image */
        for (w = 0; w < MovieWidth(movie); w++)
        {
                for (h = 0; h < MovieHeight(movie); h++)
                {
                        c = GetPixelY(entry->YUVImage, w, h) - 16;
                        d = GetPixelU(entry->YUVImage, w, h) - 128;
                        e = GetPixelV(entry->YUVImage, w, h) - 128;

                x = clip((298 * c + 409 * e + 128) >> 8);
                        SetPixelR(entry->RGBImage, w, h, x);

                        x = clip((298 * c - 100 * d - 208 * e + 128) >> 8);
                        SetPixelG(entry->RGBImage, w, h, x);

                x = clip((298 * c + 516 * d + 128) >> 8);
                        SetPixelB(entry->RGBImage, w, h, x);
                }
        }

           DeleteYUVImage(entry->YUVImage);
           entry->YUVImage = NULL;
        entry = entry->Next;
    }
}

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

```
{
    assert(movie);

    IENTRY *entry;

    entry = movie->Frames->First;

    int x = 0;
    int i = 0, w = 0, h = 0;
    int r, g, b;

    /* Loops through the frames of the movie */
    for (i = 0; i < MovieLength(movie) ; i++)
    {
       entry->YUVImage = CreateYUVImage(MovieWidth(movie), MovieHeight(movie));

       /* Setting new values to a new YUVimage */
       for (w = 0; w < MovieWidth(movie); w++)
       {
           for (h = 0; h < MovieHeight(movie); h++)
           {
              r = GetPixelR(entry->RGBImage, w, h);
              g = GetPixelG(entry->RGBImage, w, h);
              b = GetPixelB(entry->RGBImage, w, h);

              x = clip(((66 * r + 129 * g + 25 * b + 128) >> 8) + 16);
                    SetPixelY(entry->YUVImage, w, h, x);
```

```
            x = clip(((-38 * r - 74 * g + 112 * b + 128) >> 8) + 128);
                SetPixelU(entry->YUVImage, w, h, x);

            x = clip(((112 * r - 94 * g - 18 * b + 128) >> 8) + 128);
            SetPixelV(entry->YUVImage, w, h, x);
          }
        }

        DeleteImage(entry->RGBImage);
          entry->RGBImage = NULL;
          entry = entry->Next;
     }
}

int MovieLength(const MOVIE *movie)
{
   return movie->Frames->length;
}

int MovieHeight(const MOVIE *movie)
{
   if(movie->Frames->First->RGBImage)
   {
      return movie->Frames->First->RGBImage->H;
   }//if

   else if(movie->Frames->First->YUVImage)
   {
```

```
        return movie->Frames->First->YUVImage->H;
    }//if else

    else
    return 0;
}

int MovieWidth(const MOVIE *movie)
{
    if(movie->Frames->First->RGBImage)
    {
        return movie->Frames->First->RGBImage->W;
    }//if

    else if(movie->Frames->First->YUVImage)
    {
         return movie->Frames->First->YUVImage->W;
    }//if else

    else
    return 0;
}

int clip(int x)
{
     return ((x < 0) ? 0 : ((x > 255) ? 255 : x));
}
```

Movie.h

```
#ifndef MOVIE_H
#define MOVIE_H

#include "ImageList.h"

/* the movie structure */
typedef struct {
        ILIST *Frames;
} MOVIE;

/* 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);

/* Get number of frames from a movie */
int MovieLength(const MOVIE *movie);
```

```
/* Get height of movie */
int MovieHeight(const MOVIE *movie);

/* Get width of movie */
int MovieWidth(const MOVIE *movie);

#endif
```

MovieIO.c

```
#include "MovieIO.h"
#include "Constants.h"
#include "Image.h"
#include "ImageList.h"
#include "Movie.h"
#include "FileIO.h"
#include <assert.h>

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

        unsigned int i;
        MOVIE *movie = NULL;
        YUVIMAGE *YUVimage = NULL;

        movie = CreateMovie();
```

```
	if (movie == NULL) {
		return NULL;
	}

	for (i = 0; i < frameNum; i++) {
		YUVimage = LoadOneFrame(fname, i, width, height);
		if (YUVimage == NULL) {
			DeleteMovie(movie);
			return NULL;
		}

		AppendYUVImage(movie->Frames, YUVimage);
	}

	printf("The movie file %s has been read successfully!\n", fname);
	return movie;
}

/* Load one movie frame from the input file */
YUVIMAGE *LoadOneFrame(const char* fname, int n,
		unsigned int width, unsigned height)
{
	FILE *file;
	unsigned int x, y;
	unsigned char c;
	YUVIMAGE* YUVimage;

	/* Check errors */
```

```
assert(fname);
assert(n >= 0);

YUVimage = CreateYUVImage(width, height);
if (YUVimage == NULL) {
        return NULL;
}

/* Open the input file */
file = fopen(fname, "r");
if (file == NULL) {
        DeleteYUVImage(YUVimage);
        return NULL;
}

/* Find the desired frame */
fseek(file, 1.5 * n * width * height, SEEK_SET);

for (y = 0; y < height; y++) {
        for (x = 0; x < width; x++) {
                c = fgetc(file);
                SetPixelY(YUVimage, x, y, c);
        } /*rof*/
}

for (y = 0; y < height; y += 2) {
        for (x = 0; x < width; x += 2) {
                c = fgetc(file);
```

```
                    SetPixelU(YUVimage, x, y, c);
                    SetPixelU(YUVimage, x + 1, y, c);
                    SetPixelU(YUVimage, x, y + 1, c);
                    SetPixelU(YUVimage, x + 1, y + 1, c);
                }
        }

        for (y = 0; y < height; y += 2) {
                for (x = 0; x < width; x += 2) {
                    c = fgetc(file);
                    SetPixelV(YUVimage, x, y, c);
                    SetPixelV(YUVimage, x + 1, y, c);
                    SetPixelV(YUVimage, x, y + 1, c);
                    SetPixelV(YUVimage, x + 1, y + 1, c);
                }
        }

        /* Check errors */
        assert(ferror(file) == 0);

        /* Close the input file and return */
        fclose(file);
        file = NULL;
        return YUVimage;
}

/* Save the movie frames to the output file */
int SaveMovie(const char *fname, MOVIE *movie)
```

```
{
        assert(movie);
        assert(fname);

        int count;
        FILE *file;
        IENTRY *curr;

        /* Open the output file */
        file = fopen(fname, "w");
        if (file == NULL) {
                return 1;
        }

        count = 0;
        curr = movie->Frames->First;
        while (curr != NULL && movie->Frames->length > count) {

                SaveOneFrame(curr->YUVImage, fname, file);
                curr = curr->Next;
                count++;
        }

        fclose(file);
        file = NULL;

        printf("The movie file %s has been written successfully!\n", fname);
        printf("%d frames are written to the file %s in total.\n", count, fname);
```

```
        return 0;
}


/* Saves one movie frame to the output file */
void SaveOneFrame(YUVIMAGE *image, const char *fname, FILE *file)
{
        assert(image);
        assert(fname);
        assert(file);

   int x, y;
   for (y = 0; y < image->H; y++) {
      for (x = 0; x < image->W; x++) {
         fputc(GetPixelY(image, x, y), file);
      }
   }

   for (y = 0; y < image->H; y += 2) {
      for (x = 0; x < image->W; x += 2) {
         fputc(GetPixelU(image, x, y), file);
      }
   }

   for (y = 0; y < image->H; y += 2) {
      for (x = 0; x < image->W; x += 2) {
         fputc(GetPixelV(image, x, y), file);
      }
```

```
    }
}
```

MovieIO.h

```
#include "Movie.h"
#include <stdio.h>
#include <assert.h>
#include "FileIO.h"
#include "Image.h"

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

/* Load one movie frame from the input file */
YUVIMAGE *LoadOneFrame(const char* fname, int n,
                       unsigned int width, unsigned height);

/* Save the movie frames to the output file */
int SaveMovie(const char *fname, MOVIE *movie);

/* Saves one movie frame to the output file */
void SaveOneFrame(YUVIMAGE *image, const char *fname, FILE *file);
```

MovieLab.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <math.h>
#include "FileIO.h"
#include "DIPs.h"
#include "Movie.h"
#include "Constants.h"
#include "Image.h"
#include "MovieIO.h"
#include "IterativeFilter.h"

/* Menu Error Image */
void PrintImageMenu(char *ProgramName)
{
   fprintf(stderr, "\n-------------%s------------\n"
                   "Image options: \n"
                   "\t-i <file.ppm>: import image\n"
                   "\t-o <file.yuv>: export movie\n"
                   "\t-start <startVal>: set start parameter\n"
                   "\t-end <endVal>: set end parameter\n"
                   "\t-step <stepVal>: set step parameter\n"
                   "\t-hue: use hue filter\n"
                   "\t-bw: use black and white filter\n",
                   ProgramName);
}

/* Menu Error Movie */
void PrintMovieMenu(char *ProgramName)
{
   fprintf(stderr, "\n-------------%s-------------\n"
```

```
            "Movie options: \n"
            "\t-m <file.yuv>: import movie\n"
            "\t-o <file.yuv>: export movie\n"
            "\t-f <frameNum>: number of frames in the input movie\n"
            "\t-s <WxH>: size of a movie frame\n"
            "\t-reverse: use reverse filter\n",
            ProgramName);
}

int main(int argc, char *argv[])
{
    IMAGE *input = NULL;
    MOVIE *movieinput = NULL, *movieoutput = NULL;
    iterableFilter filter_func = NULL;
    char *program = NULL;
    char *inputMovie = NULL, *exportMovie = NULL, *inputImage = NULL;
    char *checkI, *checkO, *checkM;
    char *resolution, *width = NULL, *height = NULL;
    char *frame = NULL;
    int start = -1, end = -1, step;
    int importFlag = 0;
    int frameNum = 0, W = 0, H = 0;
    int reverseFlag = 0;
    int correct = 1;
    int i = 0;

    // loop through each argument for the main function
    for(int n = 0; n < argc; n++)
```

```
{
    /* Obtains program name */
    program = strtok(argv[0], "./");

  // If the user wants to import an image
  if(!strcmp(argv[n], "-i"))
  {
    /* Check if output file correct format */
    if (n != argc - 1)
    {
      checkI = strstr(argv[n+1], ".ppm");

      if (checkI)
      {
        input = LoadImage(argv[++n]);
            inputImage = argv[n];
      }
    }
    // Let the program know an image has succesfully been imported
    importFlag = 1;
  }

  // If the user wants to import a movie
  else if(!strcmp(argv[n], "-m"))
  {
    /* Check if movie input file correct format */
    if (n != argc - 1)
    {
```

```
        checkM = strstr(argv[n+1], ".yuv");

        if (checkM)
        {
            inputMovie = argv[++n];
        }
    }
    // Let the program know a movie has succesfully been imported
    importFlag = 2;
}

    /* Frame number */
    else if(!strcmp(argv[n], "-f"))
{
        if (n != argc - 1)
    {
        frame = argv[n+1];
        sscanf(frame, "%d", &frameNum);
    }
    }

/* Resolution */
    else if(!strcmp(argv[n], "-s"))
{
        /* Separates string for resolution */
    if (n != argc - 1)
    {
                resolution = strtok(argv[n+1], "x");
```

```
            width = resolution;

            while (resolution != NULL)
            {
                height = resolution;
                resolution = strtok(NULL, " ");
            }
            sscanf(width, "%d", &W);
            sscanf(height, "%d",&H);
    }
}

    /* Output file */
    else if(!strcmp(argv[n], "-o"))
{
        /* Check if output file correct format */
        if (n != argc - 1)
        {
            checkO = strstr(argv[n+1], ".yuv");

            if (checkO)
            {
                exportMovie = argv[++n];
            }
        }
}

    /* Start option */
```

```
   else if(!strcmp(argv[n], "-start"))
{
   if (n != argc - 1)
   {
      start = atoi(argv[n+1]);
   }
}

   /* End option */
   else if(!strcmp(argv[n], "-end"))
{
   if (n != argc - 1)
   {
      end = atoi(argv[n+1]);
   }
}

   /* Step option */
   else if(!strcmp(argv[n], "-step"))
{
   if (n != argc - 1)
   {
      step = atoi(argv[n+1]);
   }
}

   /* Filters */
   else if(!strcmp(argv[n], "-hue"))
```

```
        {
            filter_func = &HueRotate;
        }
        else if(!strcmp(argv[n], "-bw"))
        {
            filter_func = &BlackNWhite;
        }
        else if(!strcmp(argv[n], "-reverse"))
        {
            reverseFlag = 1;
        }

}

// Load the default image if no argument was specified
if(!importFlag)
{
    printf("\nNo -i nor -m for input file to read\n");
    return 0;
}

/* Image Option */
else if(importFlag == 1)
{
        /* Error Menu display */
    if (!exportMovie || start < 0 || end < 0 || !step || !inputImage || !filter_func)
        {
            PrintImageMenu(program);
```

```
        correct = 0;
    }
    /* Error Messages */
    if (!inputImage)
        printf("\n\tPlease provide the name of the image you want to import\n");
    if (!exportMovie)
        printf("\n\tPlease provide the name of the output file\n");
    if (start < 0)
        printf("\n\tPlease provide the start parameter\n");
    if (end < 0)
        printf("\n\tPlease provide the end parameter\n");
    if (!step)
        printf("\n\tPlease provide the step parameter\n");
if (!filter_func)
        printf("\n\tPlease provide filter\n\n");
    if (!input && inputImage)
    {
    printf("\n\tThe image file %s could not be read\n\n", inputImage);
        return 0;
    }
    if (!exportMovie || start < 0 || end < 0 || !step || !inputImage || !filter_func)
{
    DeleteImage(input);
        input = 0;
}

    if (correct == 1)
    {
```

```
            movieoutput = doIterativeFilter(input, filter_func, start, end, step);
            if (!movieoutput)
            {
                movieoutput = NULL;
            }
            else
            {
                RGB2YUVMovie(movieoutput);

                SaveMovie(exportMovie, movieoutput);
                DeleteImage(input);
                DeleteMovie(movieoutput);
            }
        }
        return 0;
}

else if(importFlag == 2)
{
    if (!exportMovie || !frameNum || !W || !H || !inputMovie)
        {
            PrintMovieMenu(program);
            correct = 0;
    }
        /* Error Messages */
        if (!inputMovie)
            printf("\n\tPlease provide the name of the movie you want to import\n");
        if (!exportMovie)
```

```
        printf("\n\tPlease provide the name of the output file\n");
        if (!frameNum)
           printf("\n\tMissing arguement for the number of frames!\n");
        if (!W || !H)
           printf("\n\tMissing arguement for the resolution of the frame!\n\n");

        if (correct == 1)
        {
           movieinput = LoadMovie(inputMovie, frameNum, W, H);

           if (!movieinput)
           {
                printf("\n\tThe movie file %s could not be read\n\n", inputMovie);
                return 0;
           }
           else
        {
                IENTRY *entry, *next;
                entry = movieinput->Frames->First;
              movieoutput = CreateMovie();
                YUVIMAGE *YUVimage = NULL;

              frameNum = (frameNum > MovieLength(movieinput) ? MovieLength(movieinput) :
frameNum);
           for (i = 1; i <= frameNum; i++)
           {
                next = entry->Next;
                   YUVimage = CopyYUVImage(entry->YUVImage);
              AppendYUVImage(movieoutput->Frames, YUVimage);
```

```
                YUVimage = NULL;
                entry = next;
            }

                if (reverseFlag)
                {
                ReverseImageList(movieoutput->Frames);
                printf("Operation Reverse is done!\n");
                }

            SaveMovie(exportMovie, movieoutput);

            DeleteMovie(movieoutput);
        }

        DeleteMovie(movieinput);
        }
        return 0;
    }
    return 0;
}
```