Advanced.c

```
#include "Advanced.h"
#include "Image.h"
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <math.h>

/* Add noise to an image */
IMAGE *Noise(IMAGE *image, int n)
{
        int x, y, i;
        int num; /* number of noise added */
        num = (n * ImageHeight(image) * ImageWidth(image)) / 100;
        srand(time(NULL));

        /* Sets the random loction of pixels to white */
        for ( i = 0; i < num; i++ )
        {
                x = ( (double)rand()/RAND_MAX )*(ImageWidth(image)-1);
                y = ( (double)rand()/RAND_MAX )*(ImageHeight(image)-1);
                SetPixelR(image, x, y, 255);
                SetPixelG(image, x, y, 255);
                SetPixelB(image, x, y, 255);
        }

        return image;
}
```

```
/* make the image posterized */
IMAGE *Posterize(IMAGE *image, int rbits, int gbits, int bbits)
{
   int x, y;
   int redOnes = 0, greenOnes = 0, blueOnes = 0;
   int i;
   int red, green, blue;

   /* Obtains ones for the lower bits */
   for (i = 1; i < rbits; i++)
   {
        redOnes += pow(2,i-1);
   }

   for (i = 1; i < gbits; i++)
   {
        greenOnes += pow(2,i-1);
   }

   for (i = 1; i < bbits; i++)
   {
        blueOnes += pow(2,i-1);
   }

   /* Loops to shift the bits */
   for (x = 0; x < ImageWidth(image); x++)
   {
        for (y = 0; y < ImageHeight(image); y++)
        {
            /* Shifts out unnecessary bits */
            red = GetPixelR(image, x, y) >> rbits;
            red = red << rbits;
            green = GetPixelG(image,x, y) >> gbits;
            green = green << gbits;
            blue = GetPixelB(image, x, y) >> bbits;
            blue = blue << bbits;

            /* Posterize */
            SetPixelR(image, x, y, (red | redOnes));
            SetPixelG(image, x, y, (green | greenOnes));
            SetPixelB(image, x, y, (blue | blueOnes));
        }
    }
    return image;
}

/* add motion blur to the image */
IMAGE *MotionBlur(IMAGE *image, int BlurAmount)
{
            int x,y,m;
            int temp_r , temp_g , temp_b;
            /* Applies Motion blur to image */
            for (x = 0; x < ImageWidth(image); x++)
            {
            for (y = ImageHeight(image) - 1; y >= 0 ; y--)
        {
            int count = 0;
            temp_r = temp_g = temp_b = 0.0;
            /* Obtains pixels values of RGB for the BlurAmount of pixels upwards of current pixel */
            for (m = 1; m<=BlurAmount ; m++)
            {
                //if ((x-m) >= 0)
                if((y-m) >= 0)
                {
                    temp_r += GetPixelR(image, x, y-m);
                    temp_b += GetPixelB(image, x, y-m);
                    temp_g += GetPixelG(image, x, y-m);
                    count++;
                }
            }
            count = (count==0)?1:count;
            /* Finds average between the original pixel and temp value fore RGB */
            SetPixelR(image, x, y, temp_r / 2.0 / count + GetPixelR(image, x, y) / 2.0);
            SetPixelB(image, x, y, temp_b / 2.0 / count + GetPixelB(image, x, y) / 2.0);
            SetPixelG(image, x, y, temp_g / 2.0 / count + GetPixelG(image, x, y) / 2.0);
        }
    }
    return image;
}

/* Enlarge image */
```

```
IMAGE *Enlarge(IMAGE *image, int percentage)
{
    double x;
    double y;
    double newX, newY;
    int duplicatex, duplicatey;
    IMAGE *E;

    double w = ImageWidth(image) * (percentage / 100.00);
    double h = ImageHeight(image) * (percentage / 100.00);
    E = CreateImage(w, h);

    /* New image is the original image */
    if (percentage == 100)
    {
        for (x = 0; x < ImageWidth(image); x++)
            {
                for (y = 0; y < ImageHeight(image); y++)
                {
                        SetPixelR(E, x, y, GetPixelR(image, x, y));
                        SetPixelG(E, x, y, GetPixelG(image, x, y));
                        SetPixelB(E, x, y, GetPixelB(image, x, y));
                }
            }
    }

    /* New image larger than original */
    else if (percentage > 100)
    {
        for (x = 0; x < ImageWidth(image); x++)
        {
            for (y = 0; y < ImageHeight(image); y++)
            {
                        newX = x * (percentage / 100.00);
                        newY = y * (percentage / 100.00);

                for (duplicatex = 0; (percentage/100.00) - duplicatex > 0; duplicatex++)
                        {
                            for (duplicatey = 0; (percentage/100.00) - duplicatey > 0; duplicatey++)
                            {
                                    SetPixelR(E, newX, newY, GetPixelR(image, x, y));
                                    SetPixelG(E, newX, newY, GetPixelG(image, x, y));
                                    SetPixelB(E, newX, newY, GetPixelB(image, x, y));

                                    newX++;
                            }
                            newX = newX - duplicatey;
                            newY++;
                        }
            }
        }
    }
    DeleteImage(image);
    image = NULL;
    return E;
}
```

```
/* Squares image */
IMAGE *Square(IMAGE *image, int x, int y, int L)
{
    int width, height;
    int maxW, maxH;
    IMAGE *sqr;
    sqr = CreateImage(L, L);

    ((x + L) > ImageWidth(image)) ? (maxW = ImageWidth(image)):(maxW = L);
    ((y + L) > ImageHeight(image)) ? (maxH = ImageHeight(image)):(maxH = L);

    for (width =  0; width < maxW; width++)
    {
            for (height = 0; height < maxH; height++)
            {
                 SetPixelR(sqr, width, height, GetPixelR(image, x + width, y + height));
             SetPixelG(sqr, width, height, GetPixelG(image, x + width, y + height));
             SetPixelB(sqr, width, height, GetPixelB(image, x + width, y + height));
            }
    }
    DeleteImage(image);
    image = NULL;
    return sqr;
}

/* Add Brightness & Constrast to image */
IMAGE *BrightnessAndContrast(IMAGE *image, int brightness, int contrast)
```

```
{
    double factor;
    int x, y;
    int r, g, b;

    /* Brightness off bounds */
    if (brightness < -255)
            brightness = -255;
    else if (brightness > 255)
            brightness = 255;

    /* Contrast off bounds */
    if (contrast < -255)
            contrast = -255;
    else if (contrast > 255)
            contrast = 255;

    /* Brightness calculations */
    for (x =  0; x < ImageWidth(image); x++)
    {
        for (y = 0; y < ImageHeight(image); y++)
            {
             SetPixelR(image, x, y, GetPixelR(image, x, y) + brightness);
             SetPixelG(image, x, y, GetPixelG(image, x, y) + brightness);
             SetPixelB(image, x, y, GetPixelB(image, x, y) + brightness);
        }
    }
```

```
/* Contrast correction factor */
factor = (double)(259 * (contrast + 255)) / (double)(255 * (259 - contrast));

/* Contrast calculations */
for (x =  0; x < ImageWidth(image); x++)
{
  for (y = 0; y < ImageHeight(image); y++)
  {
    r = (int)(factor * (GetPixelR(image, x, y) - 128) + 128);
    g = (int)(factor * (GetPixelG(image, x, y) - 128) + 128);
    b = (int)(factor * (GetPixelB(image, x, y) - 128) + 128);

    if (r > 255)
          r = 255;
      if (r < 0)
          r = 0;
    if (g > 255)
      g = 255;
    if (g < 0)
      g = 0;
    if (b > 255)
      b = 255;
    if (b < 0)
      b = 0;

    SetPixelR(image, x, y, r);
    SetPixelG(image, x, y, g);
    SetPixelB(image, x, y, b);
```

```
        }
    }

    return image;
}
```

Advanced.h

```
#ifndef ADVANCED_H_INCLUDED_
#define ADVANCED_H_INCLUDED_

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

/* add noise to an image */
IMAGE *Noise(IMAGE *image, int n);

/* posterize the image */
IMAGE *Posterize(IMAGE *image, int rbits, int gbits, int bbits);

/* motion blur */
IMAGE *MotionBlur(IMAGE *image, int BlurAmount);

/* square image */
IMAGE *Square(IMAGE *image, int x, int y, int L);

/* Enlarge image */
IMAGE *Enlarge(IMAGE *image, int percentage);
```

```
/*add Brightness & Constrast to image */
IMAGE *BrightnessAndContrast(IMAGE *image, int brightness, int contrast);

#endif /* ADVANCED_H_INCLUDED_ */
```

Constants.h

```
#ifndef CONSTANTS_H_INCLUDED_
#define CONSTANTS_H_INCLUDED_

/*** global definitions ***/
#define  WIDTH 600          /* image width */
#define  HEIGHT  400                /* image height */

#define SLEN 80             /* maximum length of file names and string*/

#define SUCCESS 0           /* return code for success */
#define EXIT 13                     /* menu item number for EXIT */
#define MAX_PIXEL 255       /* max pixel value */
#define MIN_PIXEL 0                 /* min pixel value */

#define PI 3.14159265358979323846264338327950288
#define ZOOM_FACTOR 2   /* Zooming factor for the zoom function */


#endif /* CONSTANTS_H_INCLUDED_ */
```

DIPs.c

```
#include "Advanced.h"
```

```
#include "Image.h"

#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <math.h>

/* Add noise to an image */
IMAGE *Noise(IMAGE *image, int n)
{
        int x, y, i;
        int num; /* number of noise added */
        num = (n * ImageHeight(image) * ImageWidth(image)) / 100;
        srand(time(NULL));

        /* Sets the random loction of pixels to white */
        for ( i = 0; i < num; i++ )
        {
                x = ( (double)rand()/RAND_MAX )*(ImageWidth(image)-1);
                y = ( (double)rand()/RAND_MAX )*(ImageHeight(image)-1);
                SetPixelR(image, x, y, 255);
                SetPixelG(image, x, y, 255);
                SetPixelB(image, x, y, 255);
        }

        return image;
}
```

```
/* make the image posterized */

IMAGE *Posterize(IMAGE *image, int rbits, int gbits, int bbits)

{

  int x, y;

  int redOnes = 0, greenOnes = 0, blueOnes = 0;

  int i;

  int red, green, blue;



  /* Obtains ones for the lower bits */

  for (i = 1; i < rbits; i++)

  {

     redOnes += pow(2,i-1);

  }



  for (i = 1; i < gbits; i++)

  {

     greenOnes += pow(2,i-1);

  }



  for (i = 1; i < bbits; i++)

  {

     blueOnes += pow(2,i-1);

  }



  /* Loops to shift the bits */

  for (x = 0; x < ImageWidth(image); x++)

  {

     for (y = 0; y < ImageHeight(image); y++)
```

```
    {
      /* Shifts out unnecessary bits */
      red = GetPixelR(image, x, y) >> rbits;
      red = red << rbits;
      green = GetPixelG(image,x, y) >> gbits;
      green = green << gbits;
      blue = GetPixelB(image, x, y) >> bbits;
      blue = blue << bbits;

      /* Posterize */
      SetPixelR(image, x, y, (red | redOnes));
      SetPixelG(image, x, y, (green | greenOnes));
      SetPixelB(image, x, y, (blue | blueOnes));
    }
  }
  return image;
}

/* add motion blur to the image */
IMAGE *MotionBlur(IMAGE *image, int BlurAmount)
{
      int x,y,m;
      int temp_r , temp_g , temp_b;
      /* Applies Motion blur to image */
      for (x = 0; x < ImageWidth(image); x++)
      {
      for (y = ImageHeight(image) - 1; y >= 0 ; y--)
            {
```

```
            int count = 0;
            temp_r = temp_g = temp_b = 0.0;
            /* Obtains pixels values of RGB for the BlurAmount of pixels upwards of
current pixel */
            for (m = 1; m<=BlurAmount ; m++)
            {
                //if ((x-m) >= 0)
                if((y-m) >= 0)
                {
                    temp_r += GetPixelR(image, x, y-m);
                    temp_b += GetPixelB(image, x, y-m);
                    temp_g += GetPixelG(image, x, y-m);
                    count++;
                }
            }
            count = (count==0)?1:count;
            /* Finds average between the original pixel and temp value fore RGB */
            SetPixelR(image, x, y, temp_r / 2.0 / count + GetPixelR(image, x, y) /
2.0);
            SetPixelB(image, x, y, temp_b / 2.0 / count + GetPixelB(image, x, y) /
2.0);
            SetPixelG(image, x, y, temp_g / 2.0 / count + GetPixelG(image, x, y) /
2.0);
        }
    }
    return image;
}

/* Enlarge image */
IMAGE *Enlarge(IMAGE *image, int percentage)
```

```
{
    double x;
    double y;
    double newX, newY;
    int duplicatex, duplicatey;
    IMAGE *E;

    double w = ImageWidth(image) * (percentage / 100.00);
    double h = ImageHeight(image) * (percentage / 100.00);
    E = CreateImage(w, h);

    /* New image is the original image */
    if (percentage == 100)
    {
        for (x = 0; x < ImageWidth(image); x++)
            {
                for (y = 0; y < ImageHeight(image); y++)
                {
                        SetPixelR(E, x, y, GetPixelR(image, x, y));
                        SetPixelG(E, x, y, GetPixelG(image, x, y));
                        SetPixelB(E, x, y, GetPixelB(image, x, y));
                }
            }
    }

    /* New image larger than original */
    else if (percentage > 100)
    {
```

```
    for (x = 0; x < ImageWidth(image); x++)
    {
      for (y = 0; y < ImageHeight(image); y++)
      {
            newX = x * (percentage / 100.00);
            newY = y * (percentage / 100.00);

        for (duplicatex = 0; (percentage/100.00) - duplicatex > 0; duplicatex++)
            {
              for (duplicatey = 0; (percentage/100.00) - duplicatey > 0; duplicatey++)
              {
                  SetPixelR(E, newX, newY, GetPixelR(image, x, y));
                  SetPixelG(E, newX, newY, GetPixelG(image, x, y));
                  SetPixelB(E, newX, newY, GetPixelB(image, x, y));

                  newX++;
              }
              newX = newX - duplicatey;
              newY++;
            }
      }
    }
  }
  DeleteImage(image);
  image = NULL;
  return E;
}
```

```
/* Squares image */
IMAGE *Square(IMAGE *image, int x, int y, int L)
{
    int width, height;
    int maxW, maxH;
    IMAGE *sqr;
    sqr = CreateImage(L, L);

    ((x + L) > ImageWidth(image)) ? (maxW = ImageWidth(image)):(maxW = L);
    ((y + L) > ImageHeight(image)) ? (maxH = ImageHeight(image)):(maxH = L);

    for (width =  0; width < maxW; width++)
    {
            for (height = 0; height < maxH; height++)
            {
                 SetPixelR(sqr, width, height, GetPixelR(image, x + width, y + height));
             SetPixelG(sqr, width, height, GetPixelG(image, x + width, y + height));
             SetPixelB(sqr, width, height, GetPixelB(image, x + width, y + height));
            }
    }
    DeleteImage(image);
    image = NULL;
    return sqr;
}

/* Add Brightness & Constrast to image */
IMAGE *BrightnessAndContrast(IMAGE *image, int brightness, int contrast)
{
```

```
double factor;
int x, y;
int r, g, b;

/* Brightness off bounds */
if (brightness < -255)
        brightness = -255;
else if (brightness > 255)
        brightness = 255;

/* Contrast off bounds */
if (contrast < -255)
        contrast = -255;
else if (contrast > 255)
        contrast = 255;

/* Brightness calculations */
for (x =  0; x < ImageWidth(image); x++)
{
    for (y = 0; y < ImageHeight(image); y++)
        {
         SetPixelR(image, x, y, GetPixelR(image, x, y) + brightness);
         SetPixelG(image, x, y, GetPixelG(image, x, y) + brightness);
         SetPixelB(image, x, y, GetPixelB(image, x, y) + brightness);
    }
}

/* Contrast correction factor */
```

```
factor = (double)(259 * (contrast + 255)) / (double)(255 * (259 - contrast));

/* Contrast calculations */
for (x =  0; x < ImageWidth(image); x++)
{
    for (y = 0; y < ImageHeight(image); y++)
    {
         r = (int)(factor * (GetPixelR(image, x, y) - 128) + 128);
         g = (int)(factor * (GetPixelG(image, x, y) - 128) + 128);
         b = (int)(factor * (GetPixelB(image, x, y) - 128) + 128);

         if (r > 255)
                    r = 255;
             if (r < 0)
                    r = 0;
         if (g > 255)
            g = 255;
         if (g < 0)
            g = 0;
         if (b > 255)
            b = 255;
         if (b < 0)
            b = 0;

         SetPixelR(image, x, y, r);
         SetPixelG(image, x, y, g);
         SetPixelB(image, x, y, b);
    }
    }

    return image;
}
```

DIPs.h

```
#ifndef DIPS_H_INCLUDED_
#define DIPS_H_INCLUDED_

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

/* change a color image to black & white */
IMAGE *BlackNWhite(IMAGE *image);

/* sharpen the image */
IMAGE *Sharpen(IMAGE *image);

/* change the image hue */
IMAGE *Hue(IMAGE *image, int degree);

#endif /* DIPS_H_INCLUDED_ */
```

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     ftype[] = ".ppm";
    char     fname_tmp[SLEN];
    IMAGE     *image;
    strcpy(fname_tmp, fname);
    strcat(fname_tmp, ftype);
    File = fopen(fname_tmp, "r");

    /* Unable to open file */
    if (!File)
    {
     #ifdef DEBUG
        printf("\nCan't open file \"%s\" for reading!\n", fname);
     #endif
     return NULL;
    }

    fscanf(File, "%79s", Type);
```

```
/* Unsupported file */
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);
/* Width of image out of bounds */
if (W <= 0)
{
   #ifdef DEBUG
         printf("\nUnsupported image width %d!\n", W);
   #endif
         fclose(File);
         return NULL;
}

fscanf(File, "%d", &H);
/* Height of image out of bounds */
if (H <= 0)
{
   #ifdef DEBUG
         printf("\nUnsupported image height %d!\n", H);
   #endif
```

```
            fclose(File);
            return NULL;
}

fscanf(File, "%d", &MaxValue);
/* Max Value of image not 255 */
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);
/* Error creating image */
if (!image)
{
    #ifdef DEBUG
            printf("\nError creating image from %s!\n", fname_tmp);
```

```
  #endif
    fclose(File);
    return NULL;
}
/* Retrieves the RGB of every pixel of image and reads image successfully */
else
{
    for (y = 0; y < ImageHeight(image); y++)
    {
        for (x = 0; x < ImageWidth(image); 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;
    }
}

int SaveImage(const char *fname, const IMAGE *image)
{
  assert(image != NULL && "No image to save!\n");
    FILE        *File;
    int        x, y;
    char        SysCmd[SLEN * 5];
    char        ftype[] = ".ppm";
    char        fname_tmp[SLEN];
    char        fname_tmp2[SLEN];
    unsigned int Width = ImageWidth(image);
    unsigned int Height = ImageHeight(image);
    strcpy(fname_tmp, fname);
    strcpy(fname_tmp2, fname);
    strcat(fname_tmp2, ftype);

    File = fopen(fname_tmp2, "w");
    /* Error opening file */
    if (!File)
    {
      #ifdef DEBUG
        printf("\nCan't open file \"%s\" for writing!\n", fname);
      #endif
```

```
        return 1;
}
fprintf(File, "P6\n");
fprintf(File, "%d %d\n", Width, Height);
fprintf(File, "255\n");
/* Saves RGB values of every pixel and places into file */
for (y = 0; y < Height; y++)
{
        for (x = 0; x < Width; x++)
        {
                fputc(GetPixelR(image, x, y), File);
                fputc(GetPixelG(image, x, y), File);
                fputc(GetPixelB(image, x, y), File);
        }
}
/* Error writing into file */
if (ferror(File))
{
   #ifdef DEBUG
        printf("\nError while writing to file!\n");
   #endif
        return 2;
}
fclose(File);
#ifdef DEBUG
   printf("%s was saved successfully. \n", fname_tmp2);
#endif
/*
```

```
	 * Rename file to image.ppm, convert it to ~/public_html/<fname>.jpg
	 * and make it world readable
	 */
	sprintf(SysCmd, "/users/grad2/doemer/eecs22/bin/pnmtojpeg_hw4.tcsh %s",
			fname_tmp2);
	if (system(SysCmd) != 0)
	{
	   #ifdef DEBUG
		printf("\nError while converting to JPG:\nCommand \"%s\" failed!\n", SysCmd);
	   #endif
		return 3;
	}
	#ifdef DEBUG
	   printf("%s.jpg was stored for viewing. \n", fname_tmp);
	#endif

	return 0;
}
```

FileIO.h

```
#ifndef FILE_IO_H
#define FILE_IO_H

#include "Image.h"

/* Read image from a file						*/
/* The size of the image needs to be pre-set				*/
/* The memory spaces 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);

/* Save a processed image          */
/* Return values:                  */
/* 0: successfully saved the image    */
/* 1: Cannot open the file for writing */
/* 2: File error while writing to file   */
int SaveImage(const char *fname, const IMAGE *image);

#endif
```

Image.c

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

/* Get the R intensity of pixel (x, y) in image */
unsigned char GetPixelR(const IMAGE *image, unsigned int x,  unsigned int y)
{
    assert(image);
    return image->R[x + y *image->W];
}

/* Get the G intensity of pixel (x, y) in image */
```

```
unsigned char GetPixelG(const IMAGE *image, unsigned int x,  unsigned int y)
{
    assert(image);
    return image->G[x + y *image->W];
}

/* Get the B intensity of pixel (x, y) in image */
unsigned char GetPixelB(const IMAGE *image, unsigned int x,  unsigned int y)
{
    assert(image);
    return image->B[x + y *image->W];
}

/* Set the R intensity of pixel (x, y) in image to r */
void SetPixelR(IMAGE *image, unsigned int x,  unsigned int y, unsigned char r)
{
    assert(image);
    image->R[x + y *image->W] = r;
}

/* Set the G intensity of pixel (x, y) in image to g */
void SetPixelG(IMAGE *image, unsigned int x,  unsigned int y, unsigned char g)
{
    assert(image);
    image->G[x + y *image->W] = g;
}

/* Set the B intensity of pixel (x, y) in image to b */
```

```
void SetPixelB(IMAGE *image, unsigned int x,  unsigned int y, unsigned char b)
{
    assert(image);
    image->B[x + y *image->W] = b;
}

/* Allocate dynamic memory for the image structure and its R/G/B values */
/* Return the pointer to the image, or NULL in case of error */
IMAGE *CreateImage(unsigned int Width, unsigned int Height)
{
    IMAGE *image;
    image = malloc(sizeof(IMAGE));

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

    image->W = Width;
    image->H = Height;
    image->R = NULL;
    image->G = NULL;
    image->B = NULL;

    image->R = malloc(sizeof(unsigned char) * image->W * image->H);
    image->G = malloc(sizeof(unsigned char) * image->W * image->H);
    image->B = malloc(sizeof(unsigned char) * image->W * image->H);
    return image;
}

/* Free the memory for the R/G/B values and IMAGE structure */
void DeleteImage(IMAGE *image)
{
    assert(image);
    free(image->R);
    free(image->G);
    free(image->B);
    free(image);
}

/* Return the image's width in pixels */
unsigned int ImageWidth(const IMAGE *image)
{
    assert(image);
    return image->W;

}

/* Return the image's height in pixels */
unsigned int ImageHeight(const IMAGE *image)
{
    assert(image);
    return image->H;
}
```

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;

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

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

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

/* Set the R intensity of pixel (x, y) in image to r */
void SetPixelR(IMAGE *image, unsigned int x,  unsigned int y, unsigned char r);

/* Set the G intensity of pixel (x, y) in image to g */
void SetPixelG(IMAGE *image, unsigned int x,  unsigned int y, unsigned char g);
```

```
/* Set the B intensity of pixel (x, y) in image to b */
void SetPixelB(IMAGE *image, unsigned int x,  unsigned int y, unsigned char b);

/* Allocate dynamic memory for the image structure and its R/G/B values */
/* Return the pointer to the image, or NULL in case of error */
IMAGE *CreateImage(unsigned int Width, unsigned int Height);

/* Free the memory for the R/G/B values and IMAGE structure */
void DeleteImage(IMAGE *image);

/* Return the image's width in pixels */
unsigned int ImageWidth(const IMAGE *image);

/* Return the image's height in pixels */
unsigned int ImageHeight(const IMAGE *image);

#endif
```

PhotoLab.c

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

#include "DIPs.h"
#include "Advanced.h"
#include "FileIO.h"
#include "Test.h"
#include "Image.h"
```

```
#include "Constants.h"

/*** function declarations ***/

/* print a menu */
void PrintMenu();

int main()
{
    #ifdef DEBUG
            AutoTest();
    #else
            int rc;
            int option;            /* user input option */
            char fname[SLEN];       /* input file name */
        char name[SLEN];
        IMAGE *image = NULL;
        IMAGE *result = NULL;
            rc = 1;
            PrintMenu();
            scanf("%d", &option);

            /* Hue() parameter */
            int hue;

            /* Posterize() parameter */
            unsigned char rbits, gbits, bbits;
            /* Noise() parameter */
            int n;

            /* MotionBlur() parameter */
            int blur_amount;

        /* Enlarge() parameter */
        int enlarge;

            /* Square() parameter */
            int offset_Y, offset_X, SquareSize;

            /* BrightnessAndContrast() parameter */
        int brightness, contrast;

            while (option != EXIT)
            {
                if (option == 1)
                {
                        printf("Please input the file name to load: ");
                        scanf("%s", fname);
                        image = LoadImage(fname);
                    if (image)
                           rc = SUCCESS;
                        DeleteImage(image);
                        image = NULL;
                }
            /* menu item 2 - 12 requires image is loaded first */
        else if (option >= 2 && option < 12)
        {
                    if (rc != SUCCESS)
                    {
                printf("No image to process!\n");
                    }
                    /* now image is loaded */
                    else
                    {
                switch (option)
                        {
                                case 2:
                                    if (!result)
                                            result = LoadImage(fname);
                        printf("Please input the file name to save: ");
                        scanf("%s", name);
                        SaveImage(name, result);
                        DeleteImage(result);
                                    result = NULL;
                        break;

                                case 3:
                                    image = LoadImage(fname);
                        result = BlackNWhite(image);
                        printf("\"Black amd White\" operation is done!\n");
                        break;
            case 4:

              image = LoadImage(fname);

        result = Sharpen(image);

        printf("\"Sharpen\" operation is done!\n");

        break;


            case 5:

              image = LoadImage(fname);

        printf("Please input the degree of changing hue: ");

        scanf("%d", &hue);

        result = Hue(image, hue);

        printf("\"Hue\" operation is done!\n");

              break;


            case 6:

              image = LoadImage(fname);

        printf("Please input noise percentage: ");

        scanf("%d", &n);

        result = Noise(image, n);

        printf("\"Noise\" operation is done!\n");

        break;


            case 7:

              image = LoadImage(fname);

        printf("Enter the number of posterization bits for R channel (1 to 8): ");

        scanf("%hhu", &rbits);

        printf("Enter the number of posterization bits for G channel (1 to 8): ");

        scanf("%hhu", &gbits);
```

```
                printf("Enter the number of posterization bits for B channel (1 to 8): ");
                scanf("%hhu", &bbits);
                result = Posterize(image, rbits, gbits, bbits);
                printf("\"Posterize\" operation is done!\n");
                break;

                    case 8:
                      image = LoadImage(fname);
                printf("Please input motion blur amount: ");
                scanf("%d", &blur_amount);
                result = MotionBlur(image ,blur_amount);
                printf("\"Motion Blur\" operation is done!\n");
                break;

                    case 9:
                      image = LoadImage(fname);
                      printf("Please input the enlarging percentage (integer between 100 -
200): ");
                      scanf("%d", &enlarge);
                if (enlarge < 100)
                      {
                          do
                          {
                            printf("Warning! Please input proper enlarge percentage (integer
between 100 - 200): ");
                      scanf("%d", &enlarge);

                          } while (enlarge < 100);
                      }
```

```
              result = Enlarge(image, enlarge);
                    printf("\"Enlarge the image\" operation is done!\n");
                    break;

                  case 10:
                    image = LoadImage(fname);
                    printf("Please enter the X offset value: ");
                    scanf("%d", &offset_X);
                    printf("Please enter the Y offset value: ");
                    scanf("%d", &offset_Y);
                    printf("Please input the cropped square size: ");
                    scanf("%d", &SquareSize);
                    result = Square(image, offset_X, offset_Y, SquareSize);
                    printf("\"Square\" operation is done!\n");
                    break;

                  case 11:
                    image = LoadImage(fname);
                    printf("Please input the brightness level (integer between -255 - 255):
");
                    scanf("%d", &brightness);
                    printf("Please input the contrast level (integer between -255 - 255): ");
                    scanf("%d", &contrast);
                    result = BrightnessAndContrast(image, brightness, contrast);
                    printf("\"Brightness and Contrast Adjustment\" operation is done!\n");
                    break;

              default:
            break;
```

```
                     }
                }
           }

                else if (option == 12)
           {
                     AutoTest();
                     rc = SUCCESS;    /* set returned code SUCCESS, since image is loaded */
           }

           else
           {
                     printf("Invalid selection!\n");
                }

           /* Process finished, waiting for another input */
           PrintMenu();
           scanf("%d", &option);
       }
       printf("You exit the program.\n");
    #endif

    return 0;
}


/*******************************************/
```

```
/* Function implementations should go here */
/******************************************/


/* Menu */
void PrintMenu()
{


   printf("\n------------------------------------------------\n");
   printf("1: Load a PPM image\n");
   printf("2: Save the image in PPM and JPEG format\n");
   printf("3: Change the color image to black and white\n");
   printf("4: Sharpen the image\n");
   printf("5: Change the hue of image\n");
   printf("6: Add Noise to an image\n");
   printf("7: Posterize an image\n");
   printf("8: Motion Blur\n");
   printf("9: Enlarge the picture by percentage\n");
   printf("10: Crop a square portion of the image\n");
   printf("11: Adjust the Brightness and Contrast of an image\n");
   printf("12: Test all functions\n");
   printf("13: Exit\n");
   printf("\n------------------------------------------------\n");
   printf("Please make your choice: ");
}
```

Test.c

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

```
/* Test_v2.c is updated from Test.c, due to the issue of incorrect original image shown on html */

#include "Test.h"
#include "Image.h"
#include "FileIO.h"
#include "DIPs.h"
#include "Advanced.h"

int AutoTest(void)
{
    int result;
    const char fname[SLEN] = "applestore";

    IMAGE *image = NULL;
    /* Load Image */
    image = LoadImage(fname);
    result = SaveImage("original", image);
    if (result) return result;
    #ifdef DEBUG
            printf("LoadImage & SaveImage tested!\n\n");
    #endif
    DeleteImage(image);
    image = NULL;

    /* Black & White */
    image = LoadImage(fname);
    if (! image) return 11;
    image = BlackNWhite(image);
```

```
if (! image) return 12;
result = SaveImage("bw", image);
if (result) return result;
#ifdef DEBUG
        printf("Black and White tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Sharpen */
image = LoadImage(fname);
if (! image) return 13;
image = Sharpen(image);
if (! image) return 14;
result = SaveImage("sharpen", image);
if (result) return result;
#ifdef DEBUG
        printf("Sharpen Detection tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Hue */
image = LoadImage(fname);
if (! image) return 15;
image = Hue(image, DEGREE);
if (! image) return 16;
result = SaveImage("hue", image);
```

```
if (result) return result;
#ifdef DEBUG
        printf("Hue tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Noise */
image = LoadImage(fname);
if (! image) return 17;
image = Noise(image, NOISE_PERCENTAGE);
if (! image) return 18;
result = SaveImage("noise", image);
if (result) return result;
#ifdef DEBUG
        printf("Noise tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Posterize */
image = LoadImage(fname);
if (! image) return 19;
image = Posterize(image, RBITS, GBITS, BBITS);
if (! image) return 20;
result = SaveImage("posterize", image);
if (result) return result;
#ifdef DEBUG
```

```
	printf("Posterization tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Motion Blur */
image = LoadImage(fname);
if (! image) return 21;
image = MotionBlur(image, BLURAMOUNT);
if (! image) return 22;
result = SaveImage("blur", image);
if (result) return result;
#ifdef DEBUG
	printf("MotionBlur tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Enlarge */
image = LoadImage(fname);
if (! image) return 23;
image = Enlarge(image, ENLARGE_PERCENTAGE);
if (! image) return 24;
result = SaveImage("enlarge", image);
if (result) return result;
#ifdef DEBUG
	printf("Enlarge tested!\n\n");
#endif
```

```
DeleteImage(image);
image = NULL;

/* Square */
image = LoadImage(fname);
if (! image) return 25;
image = Square(image, X_OFFSET, Y_OFFSET, SQUARE_SIZE);
if (! image) return 26;
result = SaveImage("square", image);
if (result) return result;
#ifdef DEBUG
        printf("Square tested!\n\n");
#endif
DeleteImage(image);
image = NULL;

/* Brightness & Contrast */
image = LoadImage(fname);
if (! image) return 23;
image = BrightnessAndContrast(image, BRIGHTNESS, CONTRAST);
if (! image) return 24;
result = SaveImage("brightnessandcontrast", image);
if (result) return result;
#ifdef DEBUG
        printf("Brightness and Contrast tested!\n\n");
#endif
DeleteImage(image);
image = NULL;
```

```
    return 0; /* success! */
}
```

Test.h

```
#ifndef TEST_H
#define TEST_H

/* Test all DIPs */
int AutoTest(void);

/* test parameters used in AutoTest() */

/* parameter used for Hue */
#define DEGREE 120

/* parameter used for Noise*/
#define NOISE_PERCENTAGE 30

/* parameters used for Posterize */
#define RBITS 7
#define GBITS 7
#define BBITS 7

/* parameter used for motion blur */
#define BLURAMOUNT 50

/* parameter used for enlarge */
```

```
#define ENLARGE_PERCENTAGE 170

/* parameters used for square */
#define X_OFFSET 100
#define Y_OFFSET 0
#define SQUARE_SIZE 400

/* brightness */
#define BRIGHTNESS 20
#define CONTRAST 200
#endif /* TEST_H */
```