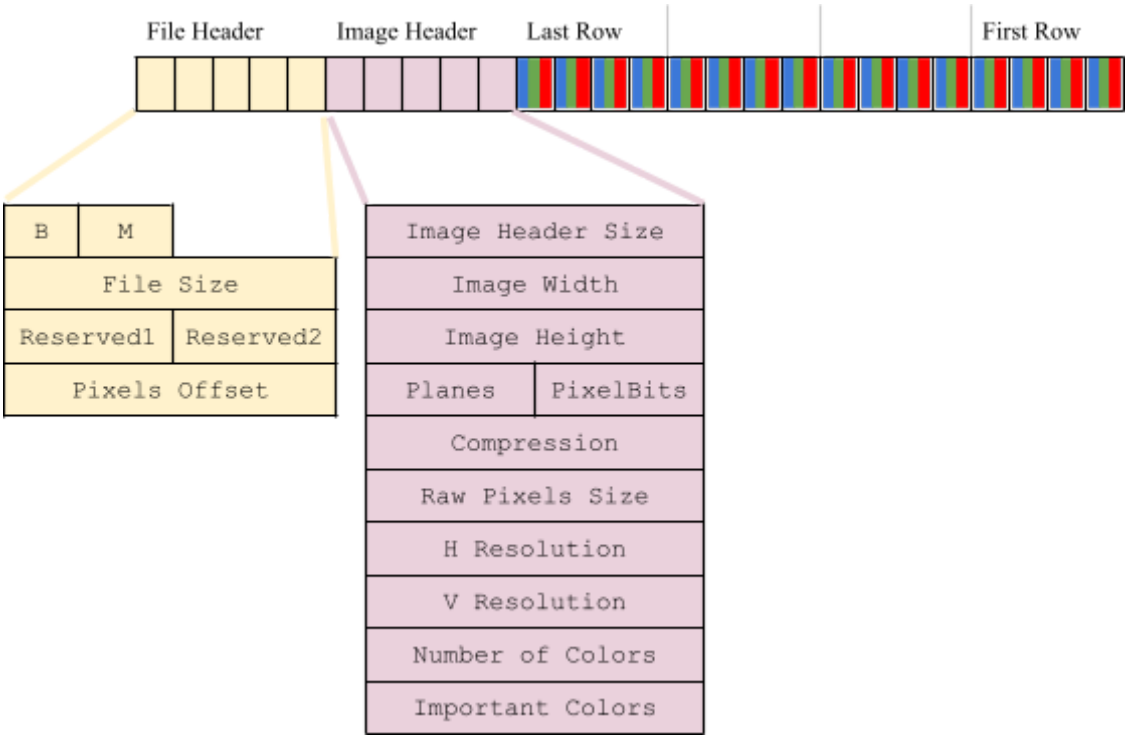

In this lab, you will write a program that loads a Bitmap Picture file, does some processing on it, and saves the resulting images back to the disk.

1. BMP file format:

In brief, the BMP file consists of 3 main parts:

1. the file header, which contains information about the file: signature (always "BM"), file size in bytes, offset value that tells where the actual pixel data starts. Some bytes are reserved for future use.
2. the image header, which contains descriptive details about the image: the header size (both headers totalling in the simple case 54 bytes), dimensions of the image (width and height in pixels), bits per pixel (how many pits are used to encode the color of each pixel), compression (method code if any), pixels data size (in bytes), printing resolutions (horizontal and vertical measured in pixels per meter), number of colors used (if not all of them) and number of important colors (usually ignored).
3. the actual pixels data, starting from the last row of the image. Inside each row, the pixel data are stored from left to right (blue, green, red components respectively). Each row ends with padding bytes that make the length of the row a multiple of 4.

2. Exercise:

1. Launch the terminal
2. Create a new directory with the name "Lab08" inside "CSC215"
3. Write a C file "mybmp.c" that:
    a. implements the function `load_image` — 2.5 points
    b. implements the function `save_image` — 2.5 points
    c. implements the function `make_grey` — 1.5 points
    d. implements the function `redify` — 1.5 points
2. Test your code using the file "test.c".

3. Assignment

Add to `mybmp.h`/`mybmp.c` a function `hmirror` that flips the image horizontally. Call your implemented function in `test.c` to test it. 2 points

mybmp.h

```
#pragma pack(2)
#if !defined MYBMP
#define MYBMP

typedef enum { cRED , cGREEN , cBLUE } Channel;

typedef struct {
  char format[2];       /* always BM */
  int file_size;        /* full size of file including headers */
  int reserved;         /* set to 0 */
  int pixel_offset;     /* where pixels data start */
} FileHead;             /* total size: 14 bytes */

typedef struct {
  int header_size;      /* set to 40 */
  int image_width;
  int image_height;
  short num_of_planes;  /* set to 1 */
  short bits_per_pixel; /* set to 24 */
  int compression;      /* set to 0 */
  int raw_pixel_size;   /* image_height*(image_width*3 + delta) */
  int h_resolution;     /* set to 2835 */
  int v_resolution;     /* set to 2835 */
  int num_of_colors;    /* set to 0 */
  int important_colors; /* set to 0 */
} ImageHead;            /* total size: 40 bytes */

typedef struct {
  unsigned char blue;
  unsigned char green;
  unsigned char red;
} Pixel; /* total size: 3 bytes */

/* opens the BMP file given be the first parameter
   returns pixels data, the file header and the image header */
Pixel** load_image(char*, FileHead*, ImageHead*);

/* takes a pixels matrix and makes its pixels grey by setting its components
   to the same value, according to the second parameter */
void make_grey(Pixel**, int, int, Channel);

/* takes a pixels matrix and modify its pixels components:
   reducing blue and grey by 50% and increasing red by (255-red)/2 */
void redify(Pixel**, int, int);

/* saves the pixels in a BMP file with the name given by the first parameter
   and using the file header and image header passed as 2nd and 3rd params */
int save_image(char* , FileHead*, ImageHead*, Pixel** );

#endif
```

create.c

```
#include <stdlib.h>
#include <stdio.h>
#include "mybmp.h"

Pixel** initialize_pixels(int h, int w){
    int i, j;
    Pixel** result = (Pixel**)malloc(h*sizeof(Pixel*));
    for (i=0; i < h; i++){
        result[i] = (Pixel*)malloc(w*sizeof(Pixel));
        for (j=0; j< w; j++){
               result[i][j].blue  = rand()%255;
               result[i][j].green  = rand()%255;
               result[i][j].red  = rand()%255;
        }
    }
    return result;
}

Pixel** initialize_pixels_linear_interpolation(int h, int w){
    int MAX_COLOR = 255;
    int i, j, sj, si, sij, x0 = 0, x1 = MAX_COLOR, y0 = 0, y1 = MAX_COLOR;
    Pixel** result = (Pixel**)malloc(h*sizeof(Pixel*));
    for (i=0; i < h; i++){
        result[i] = (Pixel*)malloc(w*sizeof(Pixel));
        si = i*MAX_COLOR/(h-1);
        for (j=0; j< w; j++){
               sj = j*MAX_COLOR/(w-1);
               sij = ((i+j)/2)*MAX_COLOR/((w+h-2)/2);
               result[i][j].red  = y0 + (sj-x0)*((y1-y0)/(x1-x0));
               result[i][j].green= y0 + (si-x0)*((y1-y0)/(x1-x0));
               result[i][j].blue = y0 + (sij-x0)*((y1-y0)/(x1-x0));
        }
    }
    return result;
}

int main(){
    FileHead fh;
    ImageHead ih;

    fh.format[0] = 'B';
    fh.format[1] = 'M';
    fh.file_size = sizeof(FileHead)+sizeof(ImageHead); /* add pixel size later*/
    fh.reserved = 0;
    fh.pixel_offset = sizeof(FileHead)+sizeof(ImageHead);

    ih.header_size = 40; /* set to 40 */
    ih.image_width = 500;
    ih.image_height = 500;
    ih.num_of_planes = 1; /* set to 1 */
    ih.bits_per_pixel = 24; /* set to 24 */
    ih.compression = 0; /* set to 0 */
    /* image_height*(image_width*3 + delta) */
    ih.raw_pixel_size = ih.image_height*(ih.image_width*3 + 0);
    ih.h_resolution = 2835; /* set to 2835 */
    ih.v_resolution = 2835; /* set to 2835 */
    ih.num_of_colors = 0; /* set to 0 */
    ih.important_colors = 0; /* set to 0 */

    fh.file_size += sizeof(Pixel)*ih.image_height*ih.image_width;

    Pixel **pixels;

    puts("initialize_pixels");
    pixels = initialize_pixels_linear_interpolation(ih.image_height, ih.image_width);

    puts("save_image");
    save_image("image.bmp", &fh, &ih, pixels);

    return 0;
}
```

test.c

```
#include <stdlib.h>
#include "mybmp.h"

Pixel** copy_pixels(Pixel** px, int h, int w){
    int i, j;
    Pixel** result = (Pixel**)malloc(h*sizeof(Pixel*));
    for (i=0; i < h; i++){
        result[i] = (Pixel*)malloc(w*sizeof(Pixel));
        for (j=0; j< w; j++) result[i][j] = px[i][j];
    }
    return result;
}

int main(){
    FileHead fh;
    ImageHead ih;
    Pixel **pixels, **grey, **red;
    pixels = load_image("image.bmp", &fh, &ih);
    grey = copy_pixels(pixels, ih.image_height, ih.image_width);
    make_grey(grey, ih.image_height, ih.image_width, cGREEN);
    save_image("image-grey.bmp", &fh, &ih, grey);
    red =copy_pixels(pixels, ih.image_height, ih.image_width);
    redify(red, ih.image_height, ih.image_width);
    save_image("image-red.bmp", &fh, &ih, red);
    return 0;
}
```