Programming Practice: Bitmap Image Processing

1. In image processing, a bitmap file is a type of file format to store digital images. The format of a bitmap file can be divided into two parts, file header and image information, as the following type declaration:

```
typedef struct {
  // File header: 14 bytes.
  char Type[2]; // Two fixed characters, "BM" for bitmap images.
  unsigned Size; // File size in bytes.
  char Reserved[4]; // Reserved field.
  unsigned OffsetBits; // Offset, i.e., the starting address of the byte where the
                          bitmap image data (pixel array) are stored
  // Image information: 40 bytes.
  unsigned InfoSize; // Information size in byte.
  unsigned Width; // Image width in pixel.
  unsigned Height; // Image height in pixel.
  unsigned short Planes; // Number of image planes in the image, must be 1.
  unsigned short BitPerPixel; // Number of bits used to represent the data for
                                 each pixel.
  unsigned Compression; // Value indicating what type of compression, if any,
                           is used, (0: uncompressed).
  unsigned ImageSize; // Size of the actual pixel data, in bytes.
  unsigned XResolution; // Preferred horizontal resolution of the image, in
                           pixels per meter.
  unsigned YResolution; // Preferred vertical resolution of the image, in pixels
                           per meter.
  unsigned Colors; // Value is zero except for indexed images using fewer than
                      the maximum number of colors.
  unsigned ImportantColors; // Number of colors that are considered important
                               when rendering the image.
} Header;
```

Following the file header and image information is the color palette with the size of OffsetBits-InfoSize-14 bytes, and then the image pixel data with the size of ImageSize bytes. The color palette and image pixel data can be declared as pointers of usigned char. The images used in this programming practice does not use color palette, i.e., OffsetBits-InfoSize-14 is 0. The pixel data area is stored using the following format:

(1) Image pixels are stored in the row-major order starting from the bottom-left corner. That is, pixels are stored from left to right and from bottom to top; the pixels in the last row are stored on the front of the pixel data area.

(2) Each pixel takes three bytes representing levels of the three original colors red, green, and blue (RGB) one byte each color. The order of the three bytes is blue color in the first byte, green in the second byte and red in the third byte.

(3) The number of bytes in a row must be a multiple of four. If the number of actual pixel bytes is not a multiple of four, 0X00's are padded at the end of the row.

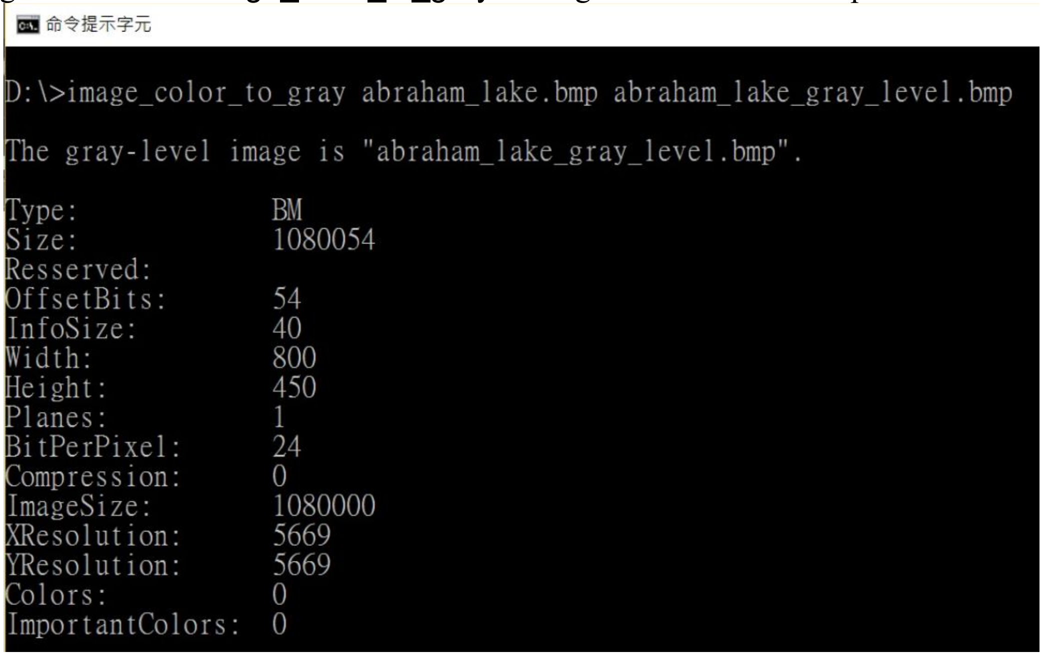
Write a C program to perform the following steps:

a. Read a color image bitmap file from disk.

b. Transform the color pixels to gray-level pixels. If the RGB level of a color pixel is of value B for blue, G for green, and R for red, the gray-level transformation formula is to set B , G , and R to (B × 28 + G × 151 + R × 77) / 256.
c. Write the gray-level image bitmap file to disk.
d. Output the resulting file header and the image information head on the screen.

Refer to Section 1.3.2 in The C Library Reference Guide for the function main: **int** main(**int** argc, **char** *argv[]) {body ...}. The default file name of the gray-level image is "gray_level.bmp". You may use example color image files blue_hills.bmp, water_lilies.bmp, abraham_lake.bmp, red_dragon.bmp, and wildflowers.bmp to test the program. If you use a color image other than the bit map format, such as png, gif, or jpg format, you may use a graphics tool to convert it to a bmp format file.

Program solution: image_color_to_gray.c. Program execution example:



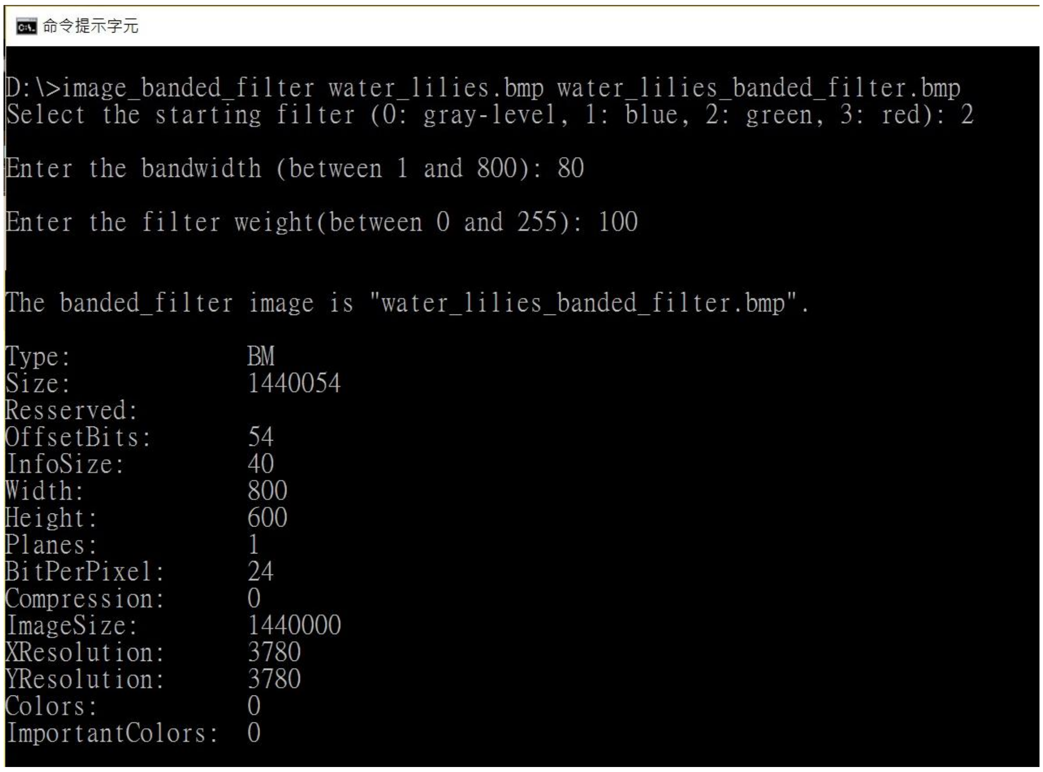
```
D:\>image_color_to_gray abraham_lake.bmp abraham_lake_gray_level.bmp

The gray-level image is "abraham_lake_gray_level.bmp".

Type:            BM
Size:            1080054
Resserved:
OffsetBits:      54
InfoSize:        40
Width:           800
Height:          450
Planes:          1
BitPerPixel:     24
Compression:     0
ImageSize:       1080000
XResolution:     5669
YResolution:     5669
Colors:          0
ImportantColors: 0
```

A color image example, abraham_lake.bmp:

The transformed gray-level image:

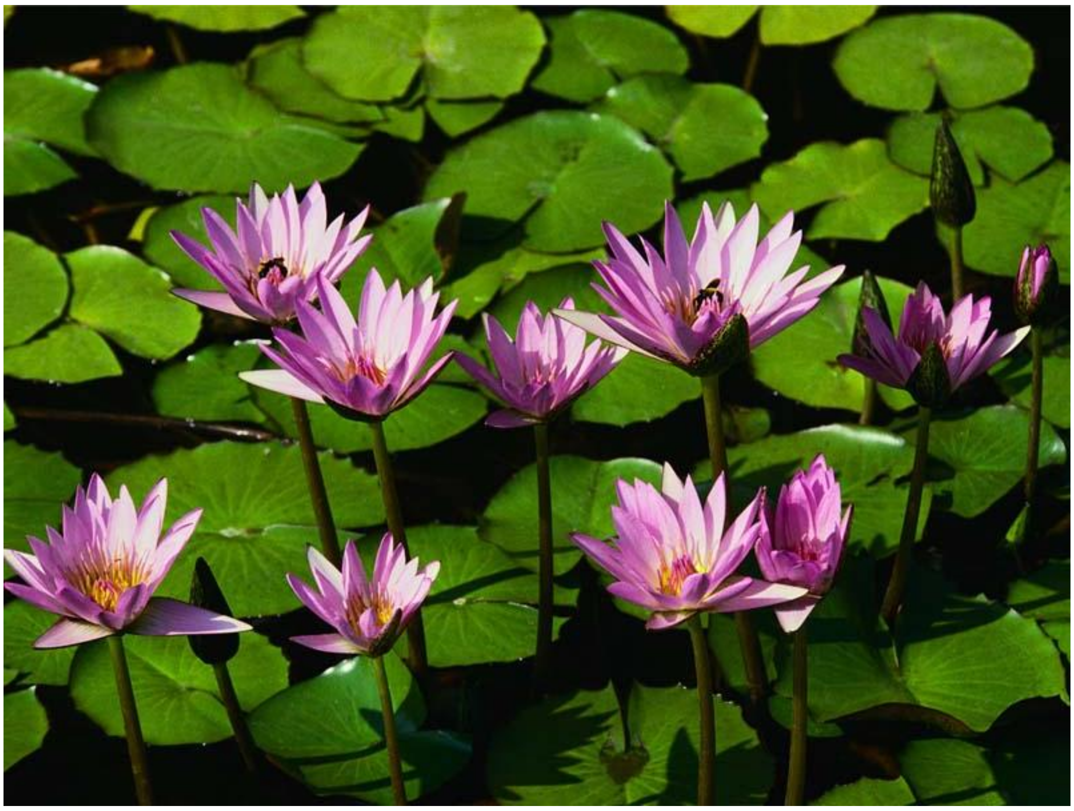
2. Write a C program to perform the following steps:
   a. Read a color image bitmap file from disk.
   b. Input three variable values: first_filter, bandwidth, weight. Variable first_filter is an integer with value between 0 and 3, denoting the starting colored filter of the bands; 0 is the gray-level filter, 1 is the blue-colored filter, 2 is the green-colored filter, and 3 is the red-colored filter. Variable bandwidth is an integer with value between 1 and the image width Header.Width. Variable weight is also an integer with value between 0 and 255 for colored filter transformation. For gray-level filter, the levels of RGB in an image pixel are set to (B × weight × 20 + G × weight × 50 + R × weight × 30) / (weight × 100); for C-colored filter, where C is level B, G, or R in an image pixel, set level of C to C<255-weight ? C+weight : weight and preserve the levels of the other two colors.
   c. Write the image bitmap file after the banded-filtering to disk.
   d. Output the resulting file header and the image information head on the screen.

Program solution: image_banded_filter.c. Program execution example:



```
D:\>image_banded_filter water_lilies.bmp water_lilies_banded_filter.bmp
Select the starting filter (0: gray-level, 1: blue, 2: green, 3: red): 2

Enter the bandwidth (between 1 and 800): 80

Enter the filter weight(between 0 and 255): 100


The banded_filter image is "water_lilies_banded_filter.bmp".

Type:             BM
Size:             1440054
Resserved:
OffsetBits:       54
InfoSize:         40
Width:            800
Height:           600
Planes:           1
BitPerPixel:      24
Compression:      0
ImageSize:        1440000
XResolution:      3780
YResolution:      3780
Colors:           0
ImportantColors:  0
```

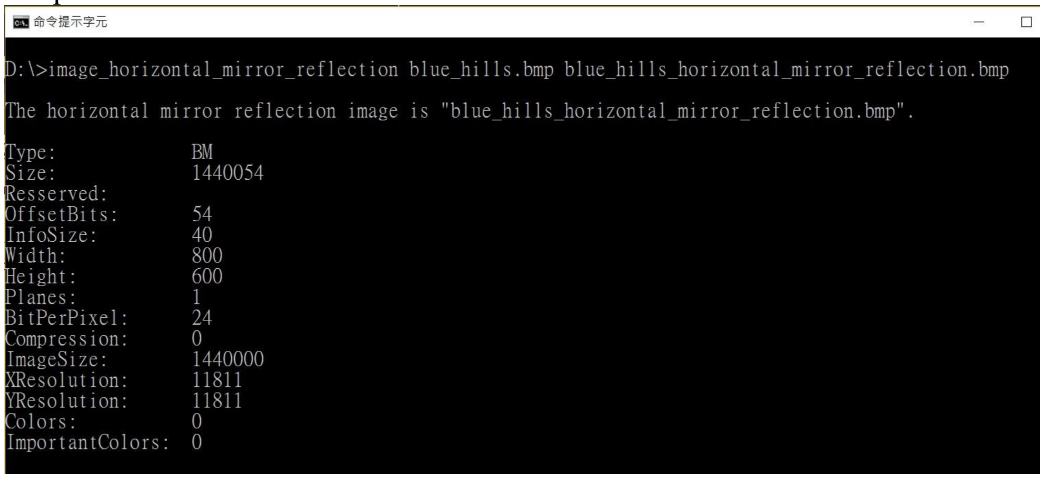
A color image example, water_lilies.bmp:



The transformed banded-filter image:

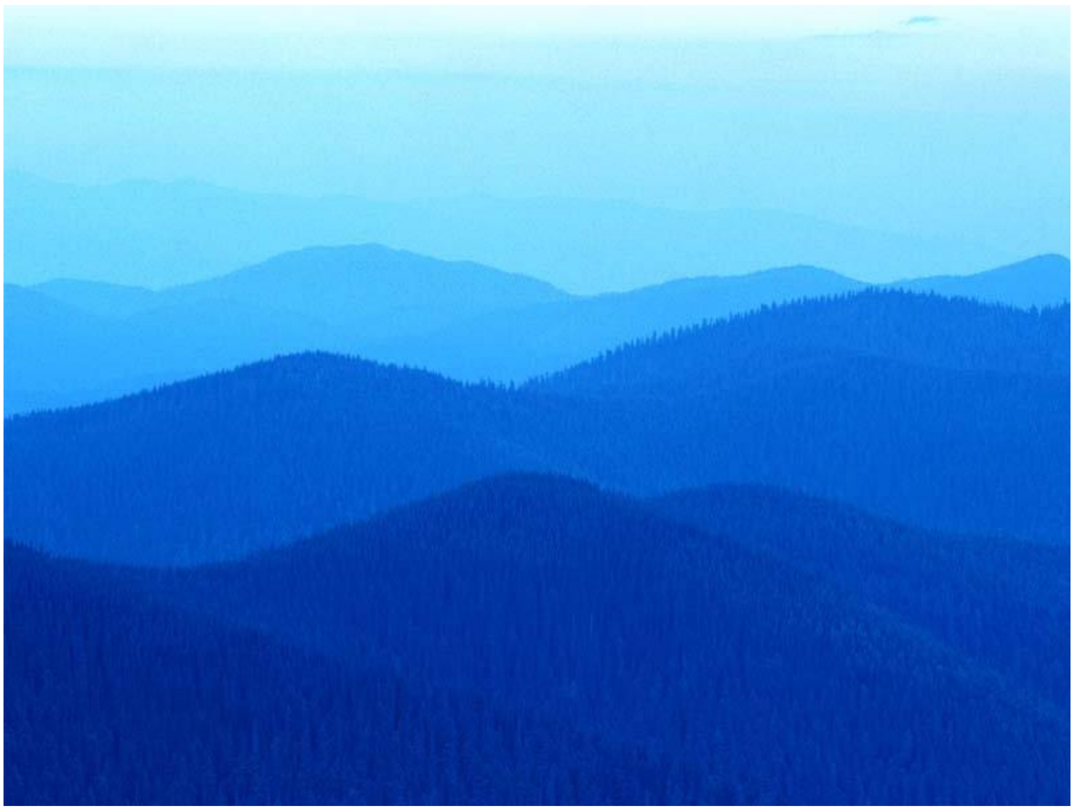
3. Write a C program to perform the following steps:
    a. Read a color image bitmap file from disk.
    b. Perform mirror reflection based on the central horizontal line of the image. That is, exchange two pixels if they are top-bottom symmetric along the central horizontal line of the image. For example, the two pixels on the top-left corner and the bottom-left corner are exchanged.
    c. Write the image bitmap file after the horizontal mirror reflection transformation to disk.
    d. Output the resulting file header and the image information head on the screen.

Program solution: image_horizontal_mirror_reflection.c. Program execution example:

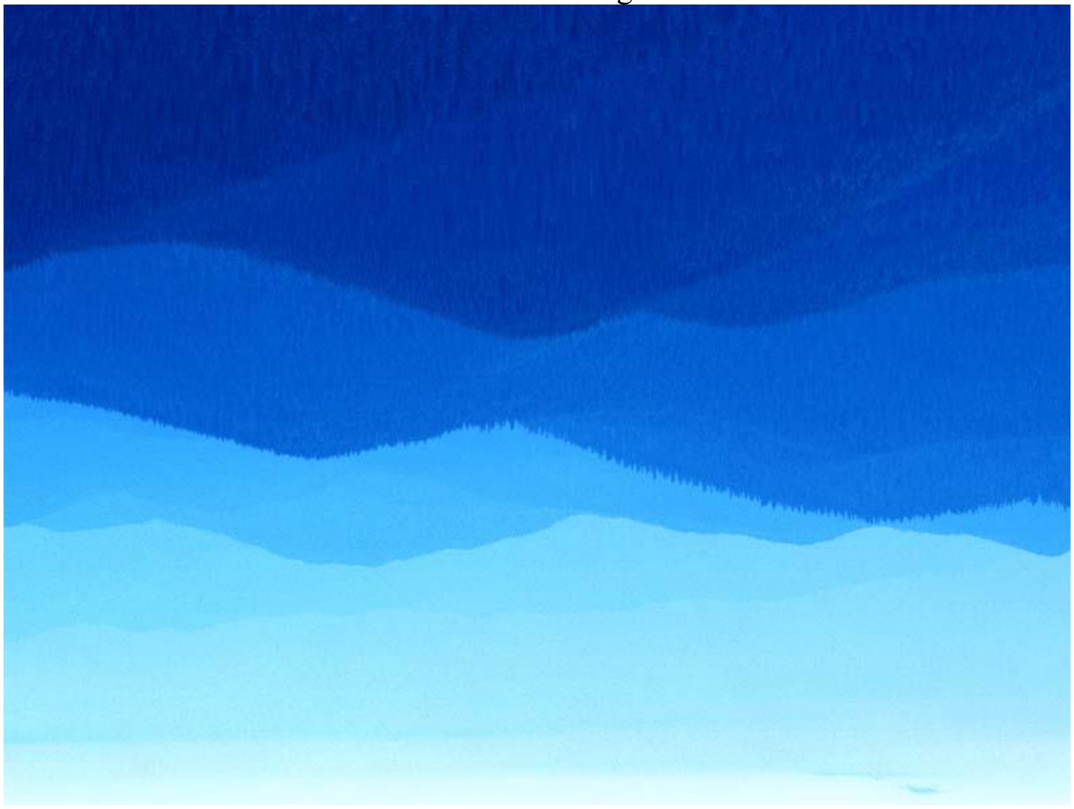


```
D:\>image_horizontal_mirror_reflection blue_hills.bmp blue_hills_horizontal_mirror_reflection.bmp

The horizontal mirror reflection image is "blue_hills_horizontal_mirror_reflection.bmp".

Type:            BM
Size:            1440054
Resserved:
OffsetBits:      54
InfoSize:        40
Width:           800
Height:          600
Planes:          1
BitPerPixel:     24
Compression:     0
ImageSize:       1440000
XResolution:     11811
YResolution:     11811
Colors:          0
ImportantColors: 0
```

A color image example, blue_hills.bmp:



The transformed horizontal mirror reflection image:



4. Image reduction to quarterly size by selecting pixels of the even rows and even columns starting from row 0 (the bottom row) and column 0 (the left-most column). For example, if a 640×800 image is reduced to one-quarter size, it becomes a 320×400 image as blocked image of Figure 1(a). An image can be transformed mirror reflection vertically and horizontally as Figure 1(b) and Figure 1(c), respectively. Also, an image can be

transformed mirror reflection centrally, as Figure 1(d), which is equivalent to a vertical mirror reflection followed by a horizontal mirror reflection or the other way around.

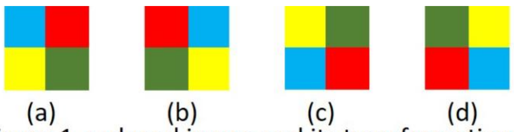


Figure 1. reduced image and its transformations

Suppose the same size of the original image is partitioned into four quadrants along the center lines in the vertical and horizontal direction. A merge transformation generates an image of the original size is to place the quarterly reduced image on the top-left corner, the 2nd quadrant as region (a) in Figure 2, the vertical mirror reflection quarterly reduced image on the top-right corner, the 1st quadrant region as (b) in Figure 2, the the horizontal mirror reflection quarterly reduced image on the bottom-left corner, the 3rd quadrant as region (c) in Figure 2, and the centrally mirror reflection quarterly reduced image on the bottom-right corner, the 4th quadrant as region (d) in Figure 2.

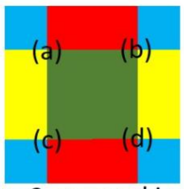
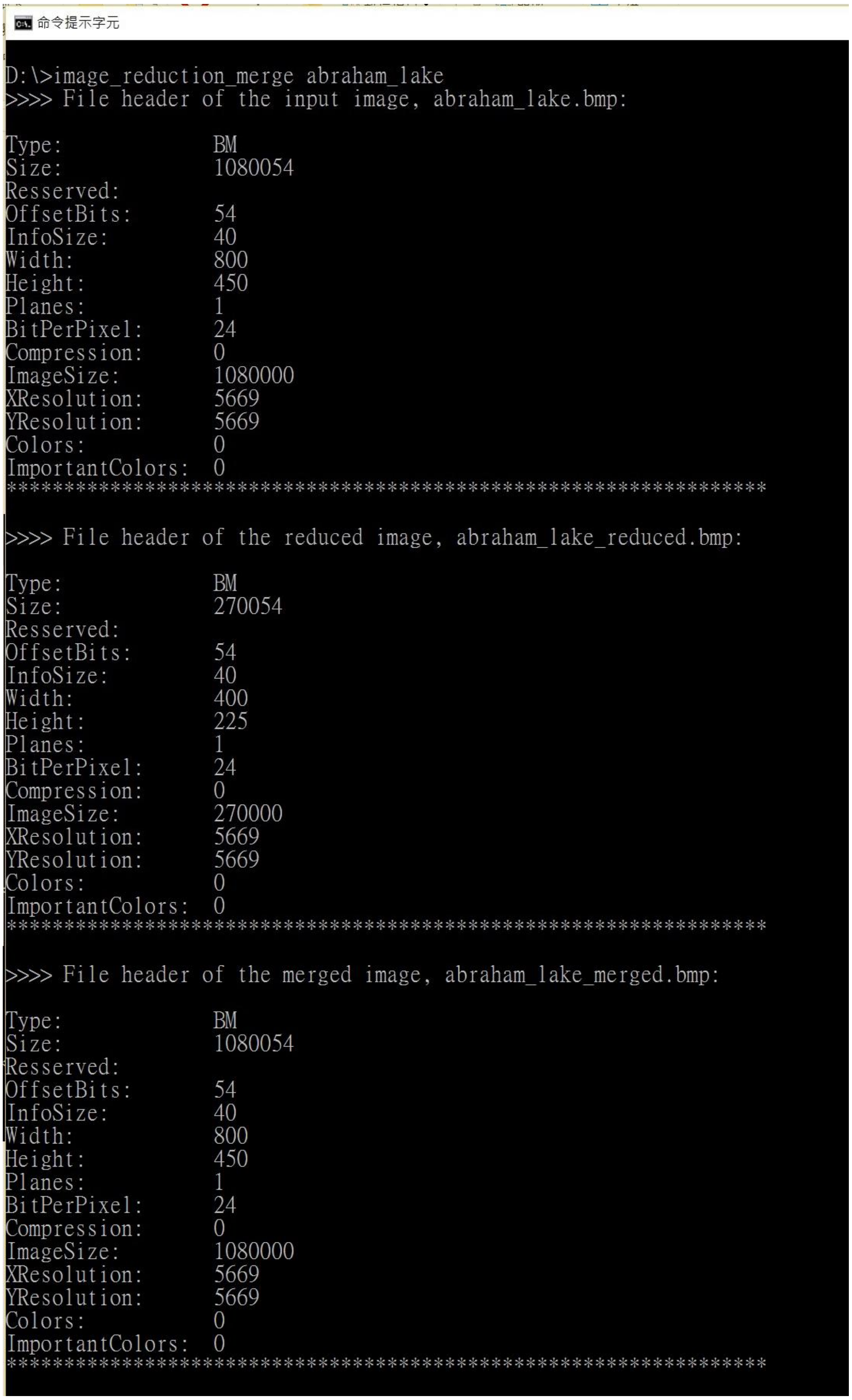


Figure 2. merged image

Write a C program to perform the following steps:

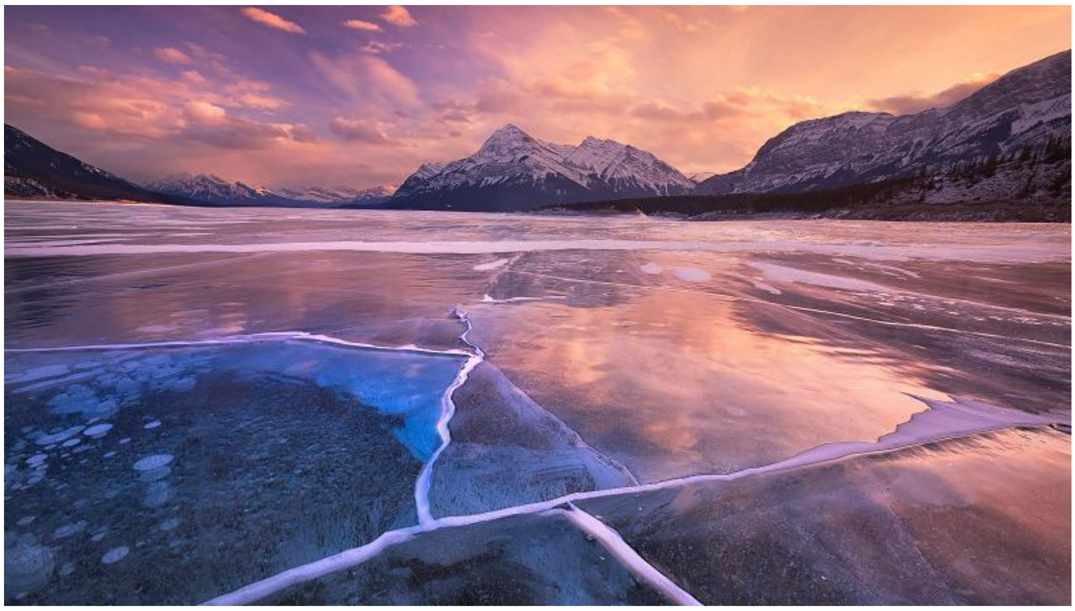
a. Read a color image bitmap file from disk.
b. Output its file header and the image information head of the input image on the screen.
c. Perform the quarterly reduction transformation.
d. Write the image bitmap file of the reduced image to disk and output its file header and the image information head on the screen. Note that some fields of the file header and the image information head of the reduced image must be modified.
e. Perform the merge transformation of the reduced image and its reflected images as in Figure 2.
f. Write the image bitmap file of the merged image to disk and output its file header and the image information head on the screen. Note that the file header and the image information head of the merged image must be identical to those of the original image.

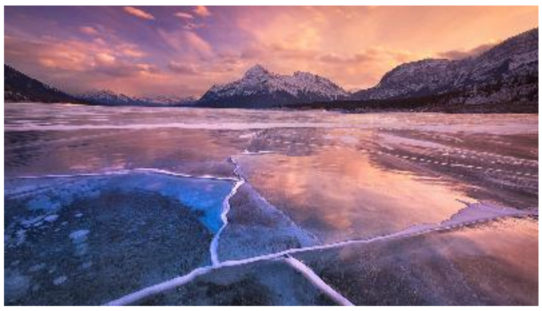
Program solution: image_horizontal_mirror_reflection.c. Program execution example:

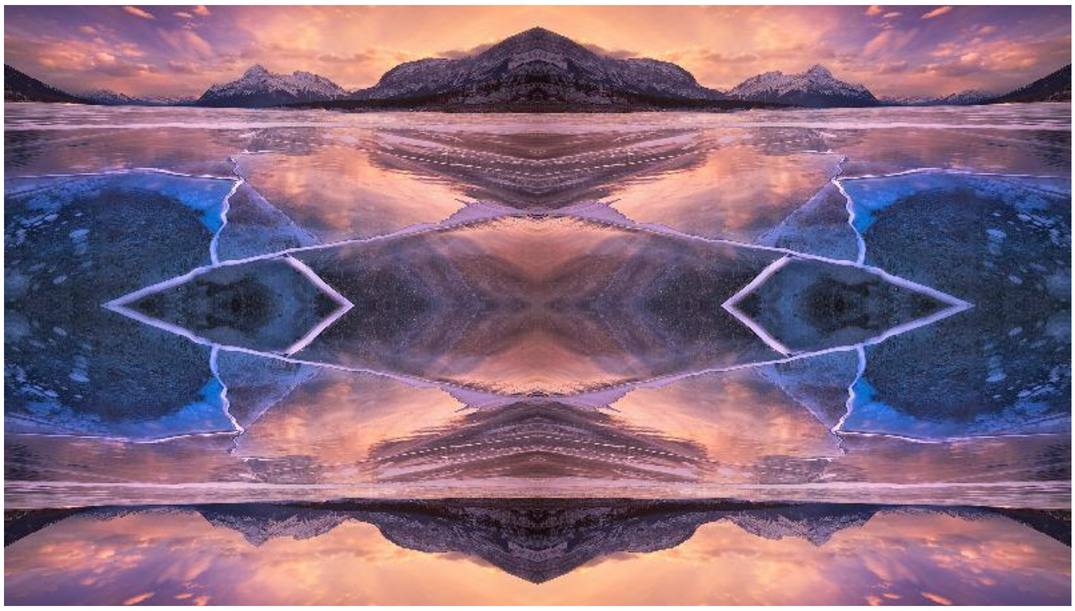


```
D:\>image_reduction_merge abraham_lake
>>>> File header of the input image, abraham_lake.bmp:

Type:               BM
Size:               1080054
Resserved:
OffsetBits:         54
InfoSize:           40
Width:              800
Height:             450
Planes:             1
BitPerPixel:        24
Compression:        0
ImageSize:          1080000
XResolution:        5669
YResolution:        5669
Colors:             0
ImportantColors:    0
*****************************************************************

>>>> File header of the reduced image, abraham_lake_reduced.bmp:

Type:               BM
Size:               270054
Resserved:
OffsetBits:         54
InfoSize:           40
Width:              400
Height:             225
Planes:             1
BitPerPixel:        24
Compression:        0
ImageSize:          270000
XResolution:        5669
YResolution:        5669
Colors:             0
ImportantColors:    0
*****************************************************************

>>>> File header of the merged image, abraham_lake_merged.bmp:

Type:               BM
Size:               1080054
Resserved:
OffsetBits:         54
InfoSize:           40
Width:              800
Height:             450
Planes:             1
BitPerPixel:        24
Compression:        0
ImageSize:          1080000
XResolution:        5669
YResolution:        5669
Colors:             0
ImportantColors:    0
*****************************************************************
```

The followings are images of an execution example.


Input image: **abraham_lake.bmp**


Reduced image: **abraham_lake_reduced.bmp**


Merged image: **abraham_lake_merged.bmp**