Programming assignment
CPE 357
Systems Programming

Program 2

Exercise:

Write a program to blend two images together in parallel.
The image files should be 24Bit per pixel (standard) BMP files. You will find several ones in the adjunct zip to this assignment. You can use your own ones – save them as 24Bit BMP in e.g. Photoshop, but line padded images MUST work.

In short:

- Blend line padded 24 Bit images from different sizes together [40%]
- Use bilinear interpolation for the pixel blending [20%]
- Blend with 1-4 processes in parallel [30%]
- Measure and time the process time correctly, with an microseconds counter [10%]

The Program:

The program should read several parameters from the command line:

[programname] [imagefile1] [imagefile2] [ratio] [process#] [outputfile]

e.g.

```
blendimages face.bmp butterfly.bmp 0.3 3 merged.bmp
```

Catch wrong or missing parameters and print a brief manual page.

The ratio should determine how much imagefile1 and imagefile2 will be in the result. A ratio of 0.5 means the resultimage will be a 50:50 mixture of both. 0.3 means 30% imagefile1 and 70% imagefile2. Got it?

So what gets blended? The pixels.
If both images have the same resolution, then the result pixel on the same x/y coordinate will be:
red_result = red_image1 * ratio + red_image2 * (1-ratio);

and so for green and blue.

The resolution of both files might be different though. In this case, the resultimage should have the same resolution as the bigger image (width). In this case, to blend the colors precisely, you need (that's a suggestion!) a function which returns the pixelcolor on a floatingpoint position:

```
unsigned char get_red(unsigned char *imagedata,float x,float y,int
imagewidth, int imageheight);
```

And use bilinear interpolation between 4 pixel, depending on the coordinates.

Howto:

First read the bitmap file header infos into corresponding structures and then the image data into an array of BYTES (unsigned char). You need to allocate this array first.
The size is dependend on the resolution, which you get with the file headers:
http://www.dragonwins.com/domains/GetTechEd/bmp/bmpfileformat.htm Wikipedia isn't bad here, but a bit chaotic:
https://en.wikipedia.org/wiki/BMP_file_format Important!

Color tables and bit masks are not necessary, that's too complex. So all optional BMP data will be skipped!
Do this for both input images!
Now allocate a BYTE array of the target size.
Loop over the bigger image resolution (in x and y) and grab the pixel color (red, green and blue). Request the pixel color of the other image as well based on the loop variables x and y. Since the other image might have a different resolution, you need to map the correct x_2 and y_2 of the other image. These values will be floats!
That means you end up grabbing a color between 4 pixels! Therefore you need to get the color of this 4 pixels and biliear calculate the real color! Blend the result color and assign it back to the result image.

Pseudo code (just to help you, if you have a better idea, follow that)

```
Read image 1 (file headers and pixel data)
Read image 2    (file headers and pixel
data) Allocate resultimage mem
Loop over the bigger one:
        Loop in x
                Loop in y
                        //Get the coords from the smaller image:
                        x_2 = … x …;
                        y_2 = … y …;
                        get the color from image 2:
                        get_red(imagedata2,x_2,y_2,…);
                        //and green, blue
                        Blend the colors
```

Assign them into the resultimage

Write a new header for the resultimageto a file

Write the pixeldata to the file

Done.

Structures for BMP Format

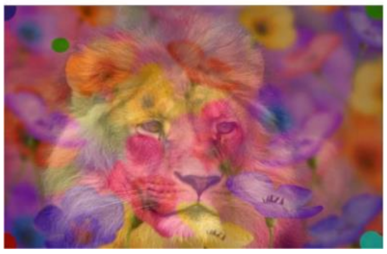
```
typedef       unsigned short
WORD; typedef unsigned int
DWORD;
typedef unsigned int  LONG;
struct tagBITMAPFILEHEADER
{
 WORD bfType; //specifies the file type
 DWORD bfSize; //specifies the size in bytes of the bitmap file
 WORD bfReserved1; //reserved; must be 0
 WORD bfReserved2; //reserved; must be 0
DWORD bfOffBits; //species the offset in bytes from the bitmapfileheader to the
bitmap bits
};
struct tagBITMAPINFOHEADER
 {
DWORD biSize; //specifies the number of bytes required by the struct
LONG biWidth; //specifies width in pixels
LONG biHeight; //species height in pixels
LORD biPlanes; //specifies the number of color planes, must be       1
WORD biBitCount; //specifies the number of bit per pixel
DWORD biCompression;//spcifies the type of compression
DWORD biSizeImage; //size of image in bytes
LONG biXPelsPerMeter; //number of pixels per meter in x axis
LONG biYPelsPerMeter; //number of pixels per meter in y axis
DWORD biClrUsed; //number of colors used by th ebitmap
DWORD biClrImportant; //number of colors that are important
};
```

Smart Start: (You do not need to follow that)

1. First read one BMP file and create a new one, saving the first one's data. That should result in an exact copy. Check if that is the case. No bilinear interpolation,
2. Do this for the second image.
3. Use two images without line padding.
   Write a get_color function which gets the data array, the width and height, the coordinates of the pixels, the color (r, g or b) and returns the color on the given xy.
   Loop over all pixels of the bigger one in x and y
    a. Grab their color in red green and blue. Do this for the second image
    b. Mix the pixels and assign them back into the target array.

   Save the file and check the image.
4. Now program a get_color_bilinear and calculate the floating point coordinates for the smaller image and try getting this done.

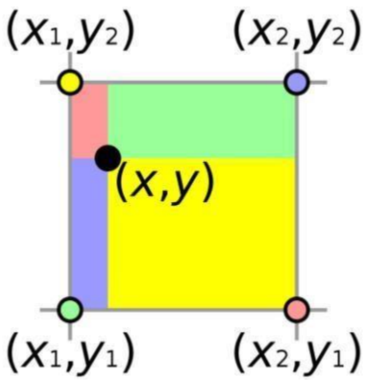
Result examples (Flower has line padding and is smaller)



Possible Hazzards:

Padding issues with reading the structures and padding issues per line of the images!

Bilinear interpolation



So you are between 4 pixels? See image, and lets say:

`x = 3.3; y = 5.8;`

consequently:

x1 = 3, x2 = 4 and y1=5, y2=6

the ratio between them is also

dx = 0.3 and dy =0.8

Then the bilinear interpolation for one color, lets say red, is:

```
Red_left_upper = get_color(…,x1,y2);
Red_right_upper = get_color(…,x2,y2);
Red_left_lower = get_color(…,x1,y1);
Red_right_lower = get_color(…,x2,y1);
```

Interpolate:

```
Red_left = Red_left_upper * (1 – dy) + Red_left_lower * dy;
Red_right = Red_right_upper * (1 – dy) + Red_right_lower * dy;
Red_result = Red_left * (1 – dx) + Red_right * dx;
```

Submit:

The whole project folder zipped as filename: FIRSTNAME_LASTNAME_PROGRAM_2.zip online. The zip must include the image files you are using, the source code file and the binary file.