Introduction to Computer Science
HW #1
Due: 2015/03/25

Homework Rules:

Hand-written homework can be handed in **before lecture starts**. Otherwise, you may contact the TA in advance and then bring the hardcopy to the TA in BL421 (please send e-mail in advance).

As for the programming part, you need to upload it to CEIBA before the deadline (2014/03/13 3am). The file you upload must be a **.zip** file that contains the following files:

README.txt

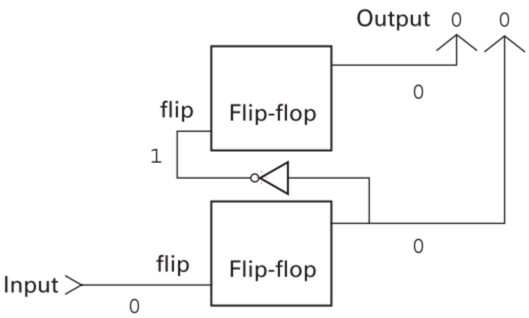
HW01_b03901XXX (a folder that contains all .cpp & .h as required),

1. Do not submit executable files (.exe) or objective files (.o, .obj). Files with names in wrong format will not be graded. You must **remove any system calls**, such as <u>system ("pause")</u>, in your code if any.
2. In README.txt, you need to describe which compiler you used in this homework and how to compile it (if it is in a “project” form).
3. In your .cpp files, we suggest you write comments as detailed as you can. If your code does not work properly, code with comments earns you more partial credits.

Introduction to Computer Science
HW #1
Due: 2015/03/25

Chapter 1 Review Problems (40%)

***3.** a. If we were to purchase a flip-flop circuit from an electronic component store, we may find that it has an additional input called *flip*. When this input changes from a 0 to 1, the output flips state (if it was 0 it is now 1 and vice versa). However, when the flip input changes from 1 to a 0, nothing happens. Even though we may not know the details of the circuitry needed to accomplish this behavior, we could still use this device as an abstract tool in other circuits. Consider the circuitry using two of the following flip-flops. If a pulse were sent on the circuit's input, the bottom flip-flop would change state. However, the second flip-flop would not change, since its input (received from the output of the NOT gate) went from a 1 to a 0. As a result, this circuit would now produce the outputs 0 and 1. A second pulse would flip the state of both flip-flops, producing an output of 1 and 0. What would be the output after a third pulse? After a fourth pulse?



***26.** Convert each of the following binary representations to its equivalent base ten representation:

a. 1111	b. 0001	c. 10101
d. 1000	e. 10011	f. 000000
g. 1001	h. 10001	i. 100001
j. 11001	k. 11010	l. 11011

***37.** Encode the following values using the 8-bit floating-point format described in Figure 1.26. Indicate each case in which a truncation error occurs.

a. $-7\frac{1}{2}$　b. $\frac{1}{2}$　c. $-3\frac{3}{4}$
d. $\frac{7}{32}$　e. $\frac{31}{32}$

***40.** What is the best approximation to the value one-tenth that can be represented using the 8-bit floating-point format described in Figure 1.26?

***45.** If you changed the length of the bit strings being used to represent integers in binary from 4 bits to 6 bits, what change would be made in the value of the largest integer you could represent? What if you were using two's complement notation?

Programming Problem (60%)

We have learned lots of data storage. Don't you ever want to know how exactly images are stored in our computer? Let's try the easiest one: bmp format. In this problem, you are going to write a **BMPImg class** that can:

(1) Load a bmp file. (**In simple format**, <u>no need to deal with arbitrarily cases</u>)
(2) Do a simple color transform: transfer the R,G,B color into gray-scale.
(But still store in R,G,B channel, we will talk about it later.)
(3) Store it as another bitmap picture. You may check it by any bmp reader.

Introduction to Computer Science
HW #1
Due: 2015/03/25

How to start? (File format)

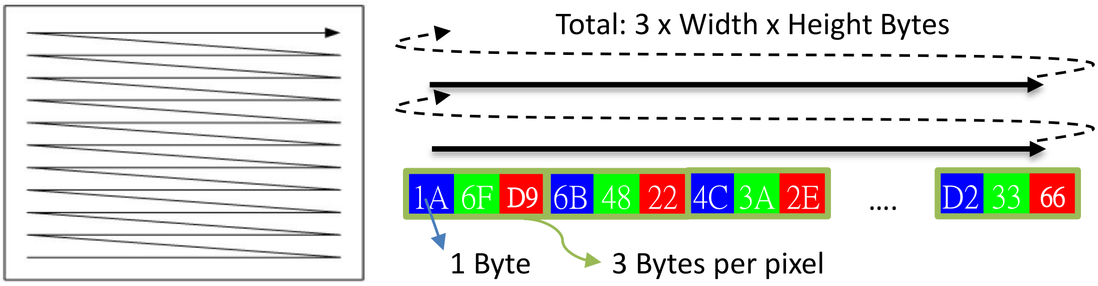
Bitmap files are composed of 2 parts: header and content (bitmap data).
The header stores a table that describes information about this picture. In this homework, we only consider the most common case as follows:

Shift	*Name*	*Size*	*Notes*
0x00	Identifier (ID)	2	Always be "BM" (char)
0x02	File Size	4	Unit: byte
0x06	Reserved	4	0
0x0A	Bitmap Data Offset	4	int(54) **in our case**
0x0E	Bitmap Header Size	4	int(40) **in our case**
0x12	Width	4	Unit: pixel
0x16	Height	4	Unit: pixel
0x1A	Planes	2	1
0x1C	Bits Per Pixel	2	24 for RGB[8,8,8] (**in our case**) 16 for RGB[5,5,5]
0x1E	Compression	4	0 **in our case** (no compression)
0x22	Bitmap Data Size	4	Unit: bytes
0x26	H-Resolution	4	Keep it untouched
0x2A	V-Resolution	4	Keep it untouched
0x2E	Used Colors	4	0 **in our case**
0x32	Important Colors	4	0 **in our case**

For more detail, you can refer to: http://crazycat1130.pixnet.net/blog/post/1345538
If you want to do it byte-by-byte yourself, be careful of the Little Endian problem.
Hint: You don't need to worry about it if you read/write as int/short directly.

As for the content, it depends on the "Bits Per Pixel" and "Compression" to determine the format. **This problem sticks with RGB24 and no-compression.** That means the color data would be stored like this:

Introduction to Computer Science
HW #1
Due: 2015/03/25

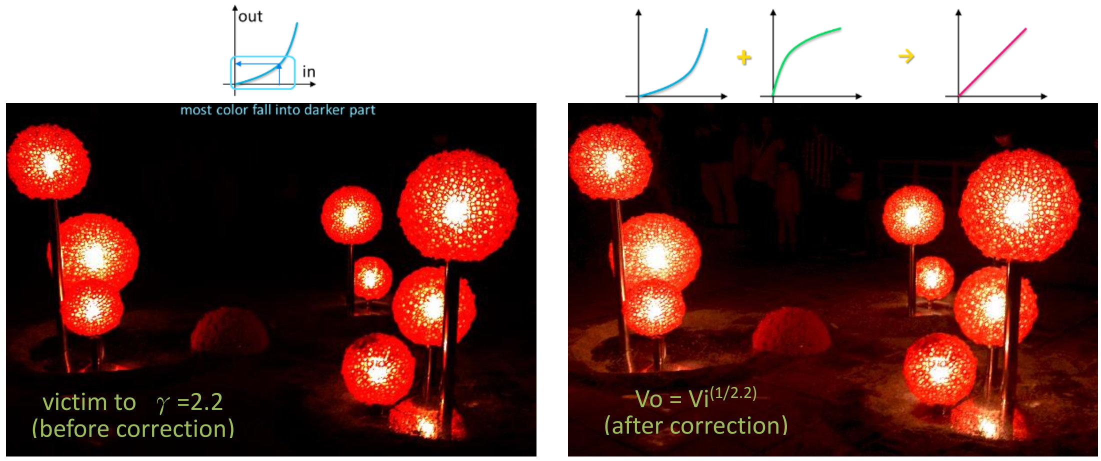


Gamma Correction:

In this homework, **you are asked to do the gamma correction** for a colored image. Gamma correction can be formulated as:

$$V_{corrected} = {V_{raw}}^{1/\gamma}, V \in [0,1], do\ this\ for\ both\ R, G, B\ channel$$

Gamma correction is a common operation in our digital cameras. Without it, pictures might be darker because the non-linear response caused by electronic sensors or display. Here's an example:



For more detail, you can refer to: http://en.wikipedia.org/wiki/Gamma_correction

A little bit inaccuracy is OK, we won't be picky.

Hint:

- "V" in the formula ranges from 0.0 to 1.0. You may need to transform unsigned char to float/double first and transform it back to 0~255 when storing it.
- Before you start, you can have a look at the code we provided. Maybe it can inspire you how to finish this problem easily. You are not requested to follow it strictly, but you need to obey the homework correcting rules below.

Introduction to Computer Science
HW #1
Due: 2015/03/25

Important rules:

You **MUST** follow these coding rules:

(1) A class named as BMPImg, TA will use it for grading.

(2) There must be these member functions as interfaces:

```
bool/void loadPic (string/char* picPath); //Loading bmp file
bool/void gammaCorrection(double gamma); //do V=V^(1/gamma)
bool/void storePic(string/char* outPath); //Store bmp file
```

(3) TA will test your code in a way like this:

```
#include "BMPImg.h"
int main(){
    BMPImg img;
    img.loadPic("lantern.bmp");
    img.storePic("result1.bmp");
    img.gammaCorrection(2.2);
    img.storePic("result2.bmp");
    return 0;
}
```

Bonus (5%): Edge Detection

Here, we are going do something special on our images. There is an easy but useful function to detect (enhance) edges in a picture! First, change the image from RGB to gray scale:

$$gray = 0.299\,\mathrm{R} + 0.587\,\mathrm{G} + 0.114\,\mathrm{B}$$

Then do the calculation:

$$G_x[x,y] =$$
$$-gray[x-1,y-1] - 2gray[x-1,y] - gray[x-1,y+1]$$
$$+gray[x+1,y-1] + 2gray[x+1,y] + gray[x+1,y+1]$$
$$G_y[x,y] =$$
$$-gray[x-1,y-1] - 2gray[x,y-1] - gray[x+1,y-1]$$
$$+gray[x-1,y+1] + 2gray[x,y+1] + gray[x+1,y+1]$$

$$Edge[x,y] = \sqrt{G_x^2 + G_y^2}$$

It's called "Sobel operator". Ref: http://en.wikipedia.org/wiki/Sobel_operator

After calculating the edge value, **store it into all R, G, and B** channels (still RGB24 format).

Introduction to Computer Science

HW #1

Due: 2015/03/25

TA will test your code this way like:

```
#include "BMPImg.h"
int main(){
    BMPImg img;
    img.loadPic("result2.bmp");
    img.sobelEdge();
    img.storePic("result3.bmp");
    return 0;
}
```

If you meet the bonus requirements, write "I finished the bonus part." in the readme file to let TA know.