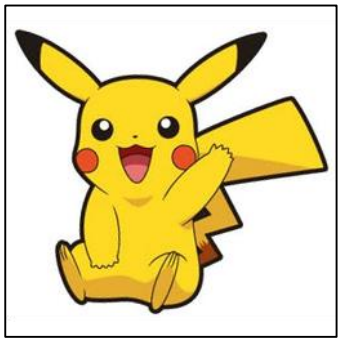
I. 作者簡介：

系級 ： 測量 112
學號 ： F64099552
姓名 ： 陳英翔
e-mail ： 0307eito@gmail.com

II. 程式簡介：

- ✓ 程式的部分，我主要是針對老師提供的範例程式碼做修改與添加
- ✓ 每一題練習題都為一個程式碼檔案。第五題的部分將「調亮」「調暗」部分各分開為兩個程式碼檔案。
- ✓ 圖檔方面，為避免額外處理 padding 的問題，採用 256x256 24 位元點陣圖

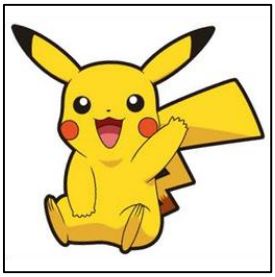


256x256

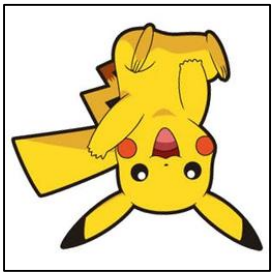
1. 將圖片旋轉 180 度並輸出

```
// Write New Image
ofstream ofl("Q1.bmp", ios::binary);
ofl.write((char*)&header, sizeof(header));
for(int i=height-1; i>=0; i--)
    for(int j=width-1; j>=0; j--)
        for(int k=0; k<bits_px/8; k++)
            ofl.write((char*)&rgb2[i][j][k], sizeof(rgb2[i][j][k]));
```

我將原本範例程式碼中，輸出新圖片的部分做修改。原本 height 和 width 是從 0 開始，低到高去堆疊，但是我將 height 和 width 這裡改為從高到低堆疊，讓最後輸出的圖片為左右顛倒、上下顛倒，符合題意。



輸入(256x256)

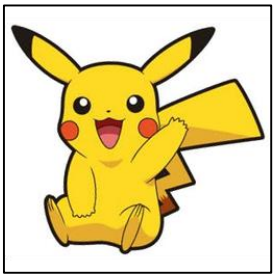


輸出(256x256)

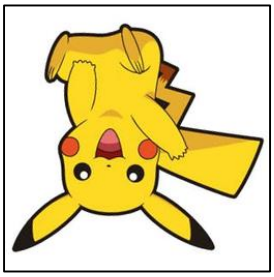
2. 將圖片垂直翻轉並輸出

```
// Write New Image
ofstream ofl("Q2.bmp", ios::binary);
ofl.write((char*)&header, sizeof(header));
for(int i=height-1; i>=0; i--)
    for(int j=0; j<width; j++)
        for(int k=0; k<bits_px/8; k++)
            ofl.write((char*)&rgb2[i][j][k], sizeof(rgb2[i][j][k]));
```

題目要求垂直翻轉，也就代表要讓圖片以上下顛倒輸出，所以只需要修改 height 的部分，將原本從低到高堆疊改為高到低堆疊，就可以得到垂直翻轉的結果。



輸入(256x256)



輸出(256x256)

3. 將圖片長寬各放大一倍並輸出

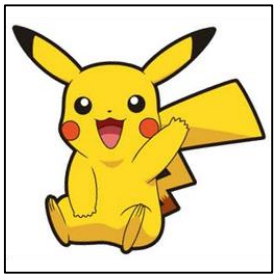
```
width = width*2;
header[18] = width % 256;
header[19] = width / 256 % 256;
header[20] = width / 256 / 256 % 256;
header[21] = width / 256 / 256 / 256 % 256;;

height = height*2;
header[22] = height &0x000000ff;
header[23] = (height >> 8) &0x000000ff;
header[24] = (height >> 16) &0x000000ff;
header[25] = (height >> 24) &0x000000ff;
```

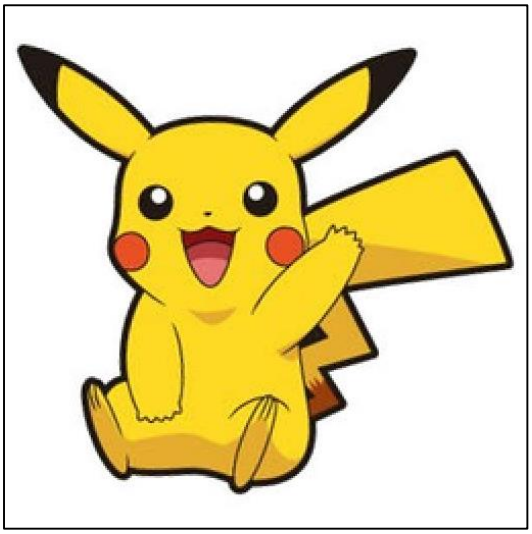
圖片的長寬要放大一倍，因此將輸出圖檔的 header 重新設定，將 width 和 height 乘 2。

```
// Write New Image
ofstream ofl("Q3.bmp", ios::binary);
ofl.write((char*)&header, sizeof(header));
for(int i=0; i<height; i++)
    for(int j=0; j<width; j++)
        for(int k=0; k<bits_px/8*2; k++)
            ofl.write((char*)&rgb2[i][j][k], sizeof(rgb2[i][j][k]));
```

輸出圖檔的部分，由於 width 和 height 在上述已經乘 2，因此這邊不須再做修改。 每一個 Pixel 的 bytes 大小必須乘 2，因為不管是垂直堆疊還是橫向堆疊的時候，width 和 height 已被拉長為兩倍，必須要用兩倍的 bytes 數才能順利完成 Pixel 的擴充。



輸入(256x256)



輸出(512x512)

4.將圖片上下兩邊裁切 1/4 高、左右兩邊裁切 1/4 寬，留下中心 1/4 面積的畫面部分並輸出

```
width = width/2;
header[18] = width % 256;
header[19] = width / 256 % 256;
header[20] = width / 256 / 256 % 256;
header[21] = width / 256 / 256 / 256 % 256;;

height = height/2;
header[22] = height &0x000000ff;
header[23] = (height >> 8) &0x000000ff;
header[24] = (height >> 16) &0x000000ff;
header[25] = (height >> 24) &0x000000ff;
```

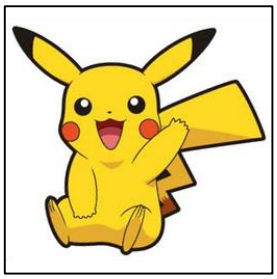
輸出的圖檔面積是原本的四分之一，因此在輸出檔的 header 中，把 width 和 height 大小都設為一半。

```
// Write New Image
ofstream ofl("Q4.bmp", ios::binary);
ofl.write((char*)&header, sizeof(header));
for(int i=0; i<height; i++)
{
    int i_temp=i+height*2/4;

    for(int j=0; j<width; j++)
    {
        int j_temp=j+width*2/4;

        for(int k=0; k<bits_px/8; k++)
        {
            ofl.write((char*)&rgb2[i_temp][j_temp][k], sizeof(rgb2[i_temp][j_temp][k]));
        }
    }
}
```

巢狀迴圈中，我設了變數 i_temp 和 j_temp，每次執行迴圈的時候，i_temp 代表 i 值加上「輸入圖檔 height 值/4」、j_temp 代表 j 值加上「輸入圖檔 width 值/4」。 之所以這麼做是因為題目要求的是圖檔中心 1/4 的面積，因此 pixel 不能從(0,0)的位置開始取，要從(height/4, width 值/4)的位置開始才對。



輸入(256x256)

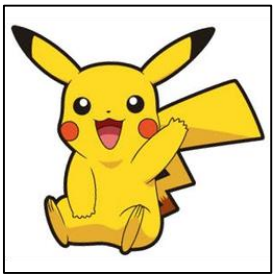
輸出(128x128)

5.調整原圖亮度(依每位同學所選的圖片，自行決定調亮或調暗)

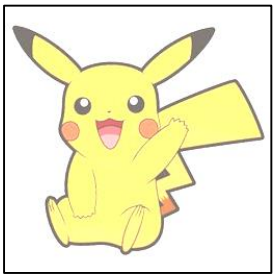
調亮

```
// Image Processing
int count=0;
unsigned char rgb2[height][width][bits_px/8];
for(int i=0; i<height; i++)
{
    for(int j=0; j<width; j++)
    {
        for(int k=0; k<bits_px/8; k++)
        {
            if ((int)rgb[i][j][k]+100>255)
            {
                rgb2[i][j][k]=255;
            }
            else
            {
                rgb2[i][j][k] = rgb[i][j][k]+100;
            }
        }
    }
}
```

在處理讀取出來的 data 的 rgb 值的時候，順便把 rgb 三原色的值分都加上 100，調高整體亮度。rgb 三個值的最大是 255，因此需要設定 if 敘述，將 255 設為上限。



輸入(256x256)

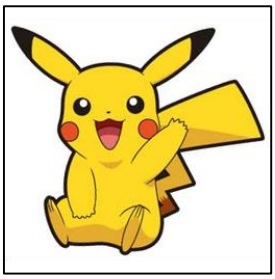


輸出(256x256)

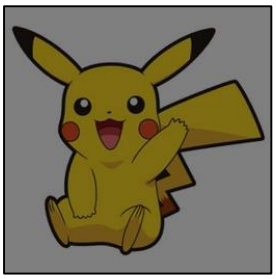
調暗

```
// Image Processing
int count=0;
unsigned char rgb2[height][width][bits_px/8];
for(int i=0; i<height; i++)
{
    for(int j=0; j<width; j++)
    {
        for(int k=0; k<bits_px/8; k++)
        {
            rgb2[i][j][k] = rgb[i][j][k]/2;
        }
    }
}
```

調暗的設定上，我把 rgb 的值乘 1/2，由於 rgb 值的最小值是 0，而乘上 1/2 的動作並不會使值小於 0，因此不需要額外假設值小於 0 的情況。



輸入(256x256)



輸出(256x256)

Histogram equalization 直方圖均衡化

原理

目的是將原始圖像像素的色彩強度均勻地映射到整個色彩範圍內，得到一個色彩強度分佈均勻的圖像。

計算舉例

嘗試將 5x5 圖像的色彩範圍均衡化至更大的色彩範圍。

假設一個 5x5 像素的圖像，色彩範圍設為[0,6]

1	4	2	4	4
5	0	4	6	6
1	0	3	1	6
6	5	6	6	0
0	5	3	4	4

統計每個色彩值的出現次數

色彩值	0	1	2	3	4	5	6
出現次數	4	3	1	2	6	3	6

計算每個色彩值的出現機率、累積機率

出現機率	4/25	3/25	1/25	2/25	6/25	3/25	6/25
	=0.16	=0.12	=0.04	=0.08	=0.24	=0.12	=0.24
累積機率	0.16	0.28	0.32	0.4	0.64	0.76	1.0

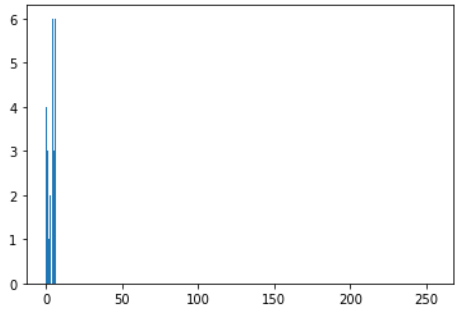
擴大色彩範圍至[0,255]，做均衡化的動作

色彩值	0	1	2	3	4	5	6
出現次數	4	3	1	2	6	3	6
累積機率	0.16	0.28	0.32	0.4	0.64	0.76	1.0
新的色彩值	0.16x255	0.28x255	0.32x255	0.4x255	0.64x255	0.76x255	1.0x255
	≒41	=71	≒82	=102	≒163	≒194	=255

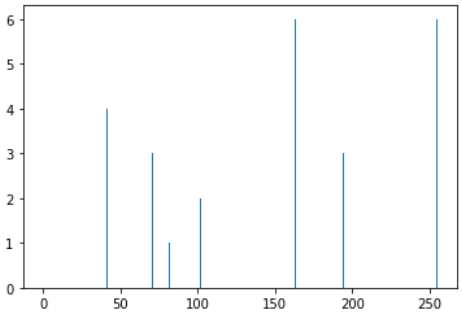
均衡化之後的結果

新的色彩值	41	71	82	102	163	194	255
出現次數	4	3	1	2	6	3	6

嘗試將分佈視覺化(使用 matplotlib)



均衡化前



均衡化後