程式設計(二)

期末專題

1102065

游竣捷

目錄

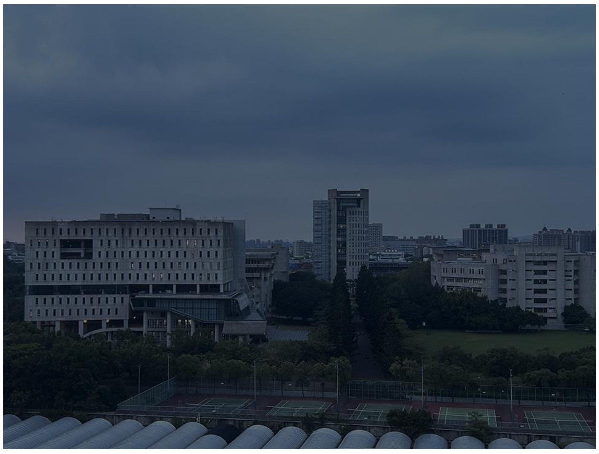
程式說明

此程式為本學期上課內容作為延伸，原先功能已包含讀取和寫入 Bitmap，將 Bitmap 圖片轉換為黑白、做模糊化、二值化、使用遮罩產生描邊效果、兩張 Bitmap 圖片相加做合成。

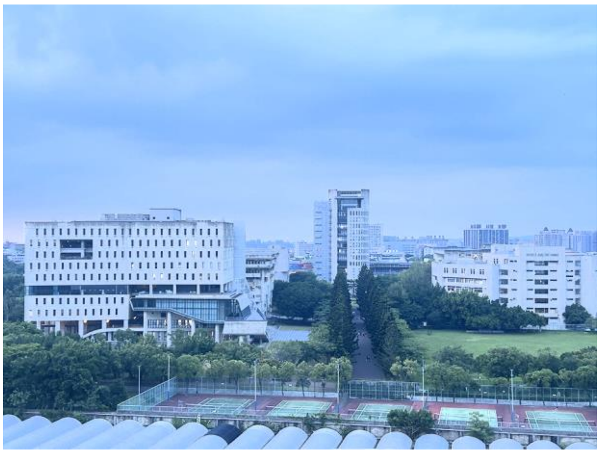
延伸功能

可自由調整 Bitmap 的整體亮度，如下圖所示。

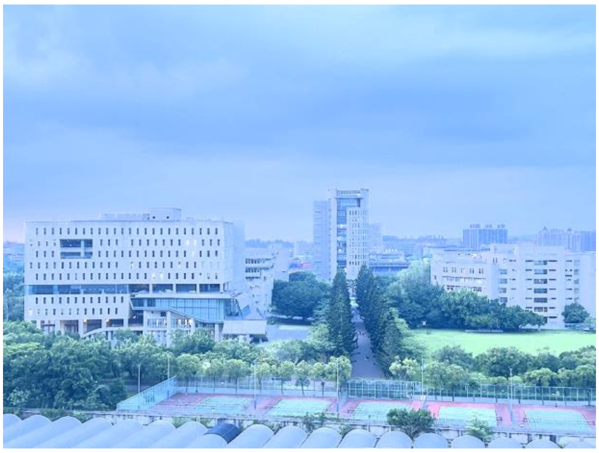
原圖



亮度調暗



亮度調亮



亮度更亮

系統分析

1. 讀取的 Bitmap 資料色彩空間為 RGB。
2. 將 Bitmap 資料的每一個像素的 RGB 透過公式轉換為 HSV。
3. 透過將每個像素的 HSV 的 V 值乘以一個倍數，調整整體圖像的亮度。
4. 將改變後的 HSV 數值轉換回 RGB。
5.

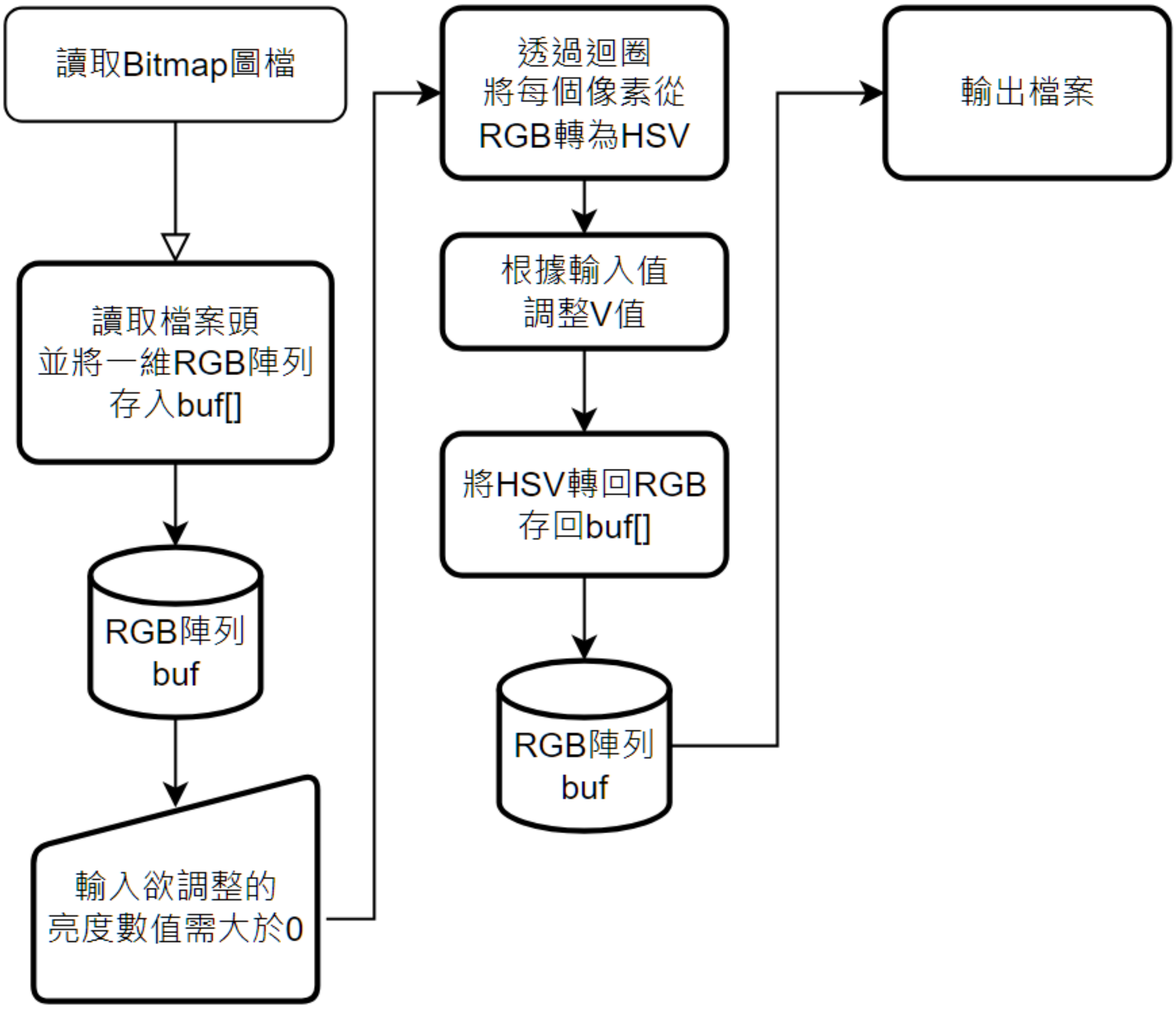
注意:

1. RGB 範圍為 0~255 之間
2. H 範圍為 0~360 之間
3. S 和 V 範圍為 0~1 之間

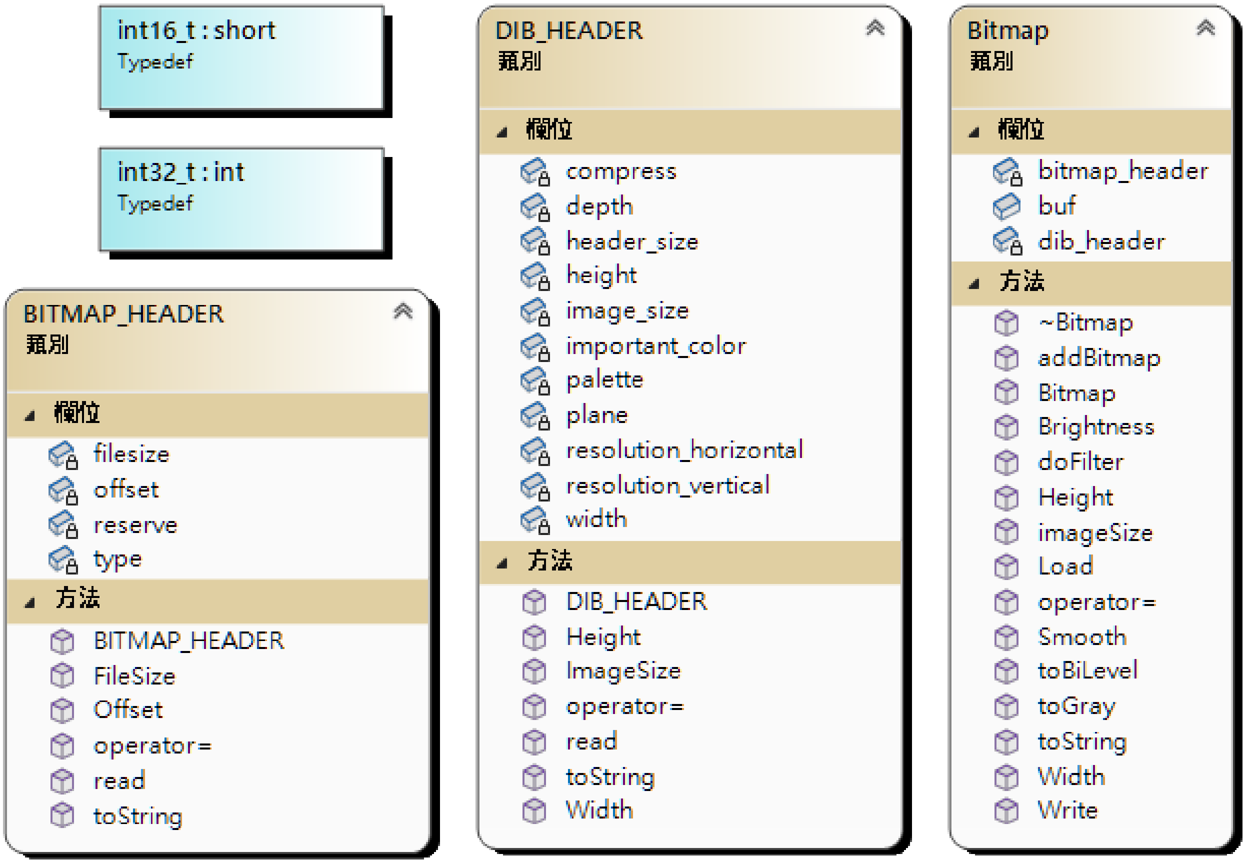
轉換公式參考自維基百科

https://en.wikipedia.org/wiki/HSL_and_HSV

流程圖



類別圖


int16_t : short
Typedef
int32_t : int
Typedef
BITMAP_HEADER
類別
欄位
filesize
offset
reserve
type
方法
BITMAP_HEADER
FileSize
Offset
operator=
read
toString
DIB_HEADER
類別
欄位
compress
depth
header_size
height
image_size
important_color
palette
plane
resolution_horizontal
resolution_vertical
width
方法
DIB_HEADER
Height
ImageSize
operator=
read
toString
Width
Bitmap
類別
欄位
bitmap_header
buf
dib_header
方法
~Bitmap
addBitmap
Bitmap
Brightness
doFilter
Height
imageSize
Load
operator=
Smooth
toBiLevel
toGray
toString
Width
Write

主要程式碼說明

```
/*****************************************************************
    Value needs to be a float num greater than zero,
    0~1 will decrease brightness, 1~2 will increase brightness.
    if Value is too bigger, the result will be brighter,
    but it maybe isn't a good result.
    輸入的數值為大於0的浮點數，
    0 ~ 1(ex. 0.5) 會將低圖像的亮度
    1 ~ 2(ex. 1.5) 會增加圖像的亮度
*****************************************************************/

    void Brightness(double Value) {
        int rawsize = imageSize(); //Bitmap圖像RGB陣列長度
        int width = Width(); //圖像寬度
        int height = Height();//圖像高度
        int BitsPerPixel = 24; //色彩深度
        //重新計算Rowsize使其適配所有影像大小
        int RowSize = ((BitsPerPixel * width + 31) / 32) * 4;
        int i;

        //透過迴圈讀取每一個像素點的RGB
        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                //透過長寬計算像素點在一維陣列中的相對位置
                i = y * RowSize + (x * 3) % RowSize;

                // 讀取 RGB 值
                // buf中的數值有可能為負數。所以資料型別使用
                // unsigned char
                unsigned char b = buf[i];
                unsigned char g = buf[i + 1];
                unsigned char r = buf[i + 2];
```

```
// RGB To HSV Formula
// 參考致維基百科公式
// 正規化 使RGB數值為 0~1 之間
// 將unsigned char 轉為 double / 255  做正規化
double R = static_cast<double>(r) / 255;
double G = static_cast<double>(g) / 255;
double B = static_cast<double>(b) / 255;

double H, S, V;
// 取得 RGB 中最大與最小的數值，並計算兩者的差值，
// 用於轉換公式
double Max = max(max(R, G), B);
double Min = min(min(R, G), B);
double Delta = Max - Min;

// Define H
if (Delta == 0) { //如果最大值與最小值相等，即差值為0
    H = 0;  // H = 0
}
else if (Max == R) { //如果最大值是 R，帶入相對應公式
    // fmod()為浮點數取餘數的函式
    H = 60 * (fmod(((G - B) / Delta), 6));
}
else if (Max == G) { //如果最大值是 G，帶入相對應公式
    H = 60 * ((B - R) / Delta + 2);
}
else if (Max == B) { //如果最大值是 B，帶入相對應公式
    H = 60 * ((R - G) / Delta + 4);
}

//執行過程中發現計算後的H值有可能為負的，
//而色調 H 為360度色環，因此加上360度修正。
if (H < 0) {
    H += 360;
}
```

```
// Define S
if (Max == 0) { //如果最大值等於0
    S = 0;  //飽和度 S = 0
}
else {  //否則，帶入對應的公式
    S = Delta / Max;
}
// Define V
V = Max; // V值對於最大值
// END of RGB To HSV Formula

// Adjust V's Value
// 調整V的數值，須注意的是V的大小為 0~1之間
// 所以這邊使用max()、min()，讓 V*乘以輸入的數值後，
// V 的大小仍在0~1之間
V = max(0.0,min(V * Value, 1.0));


// HSV To RGB Formula
// 依照公式表計算欲轉換至RGB所需的數值
double C = V * S;
double h = fmod(H / 60, 6);
double X = C * (1 - abs(fmod(h, 2) - 1));
double m = V - C;
r = g = b = 0;

// h取完餘數後，大小為0~6之間的浮點數
// 參照公式，根據h值的不同，會有不同的RGB給值組合
// 因為之前正規化除以255，所以現在將255乘回去
if (0 <= h && h < 1) {
    r = (C + m) * 255;
    g = (X + m) * 255;
    b = (m) * 255;
}
else if (1 <= h && h < 2) {
    r = (X + m) * 255;
    g = (C + m) * 255;
```

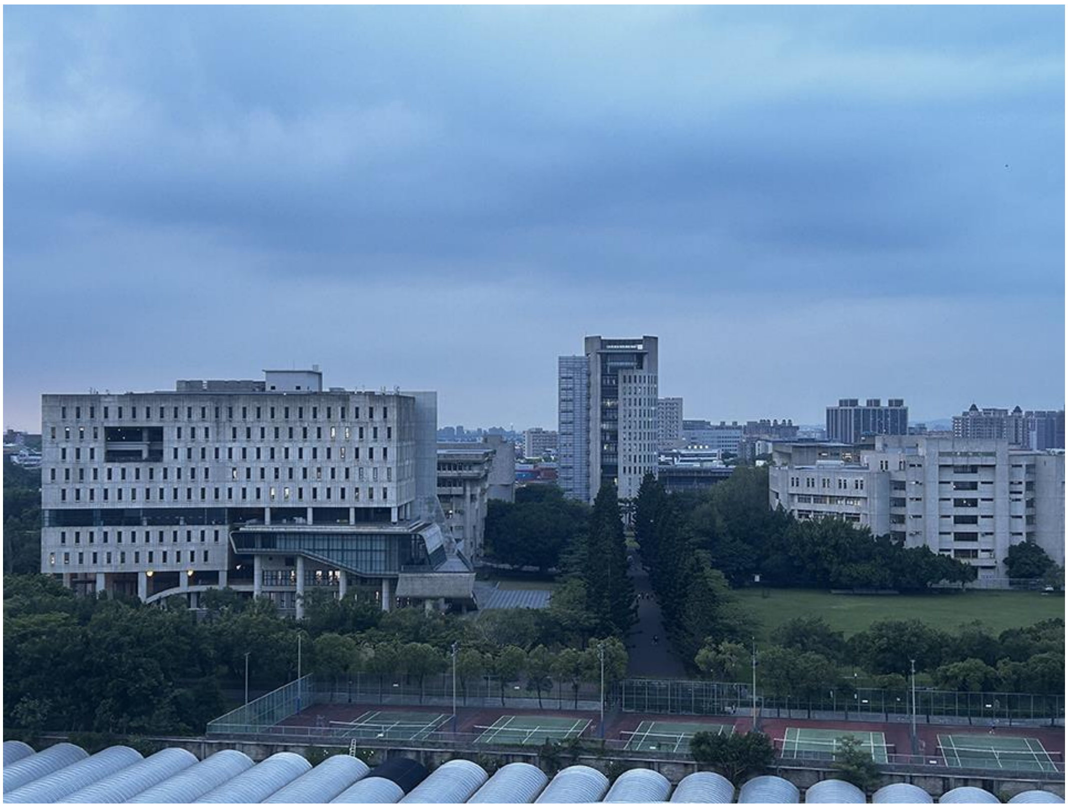
```
                b = (m) * 255;
            }
            else if (2 <= h && h < 3) {
                r = (m) * 255;
                g = (C + m) * 255;
                b = (X + m) * 255;
            }
            else if (3 <= h && h < 4) {
                r = (m) * 255;
                g = (X + m) * 255;
                b = (C + m) * 255;
            }
            else if (4 <= h && h < 5) {
                r = (X + m) * 255;
                g = (m) * 255;
                b = (C + m) * 255;
            }
            else if (5 <= h && h < 6) {
                r = (C + m) * 255;
                g = (m) * 255;
                b = (X + m) * 255;
            }
            // END of HSV To RGB Formula

            // Store RGB to buf[]
            //將HSV轉換至RGB的值存至 buf[] 中
            buf[i] = b;
            buf[i + 1] = g;
            buf[i + 2] = r;
        }
    }
}
```

測試

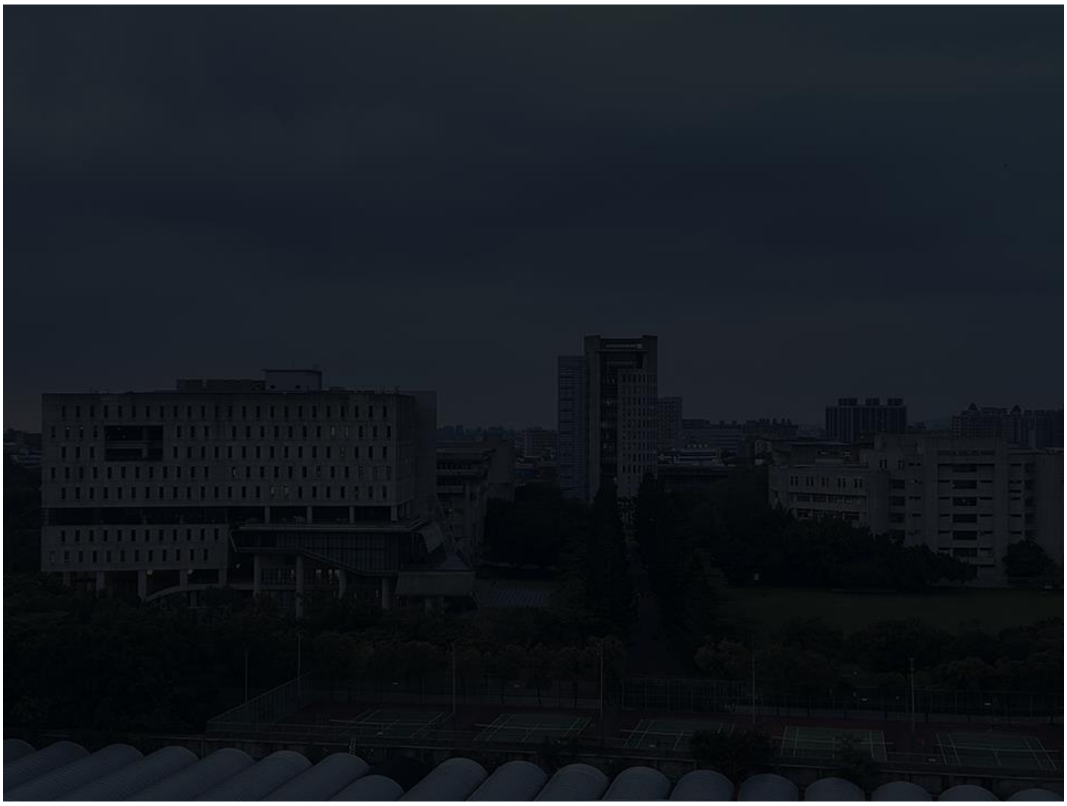
測試程式碼

```
void make_dir(const string dir_name)
{
    if (_access(dir_name.c_str(), 0) == -1)//如果路徑不存在，無法存取
        int re = _mkdir(dir_name.c_str());//建立資料夾
}

void BrightTest() {
    Bitmap bmp, bmp2, bmp3, bmp4, bmp5;

    string path = "BrightTest";
    make_dir(path); //如果路徑不存在，建立資料夾

    bmp.Load("campus.bmp"); //讀檔
    bmp2 = bmp3 = bmp4 = bmp5 = bmp;

    bmp.Brightness(0.2); //數值0.2，亮度變非常暗
    bmp.Write(path + "/result1.bmp");

    bmp2.Brightness(0.5); //數值0.5，亮度變暗
    bmp2.Write(path + "/result2.bmp");

    bmp3.Brightness(1.5); //數值1.5，亮度變亮
    bmp3.Write(path + "/result3.bmp");

    bmp4.Brightness(2.5); //數值2.5，亮度變很亮
    bmp4.Write(path + "/result4.bmp");

    bmp5.Brightness(4);   //數值4，亮度變非常亮
    bmp5.Write(path + "/result5.bmp");
}
```

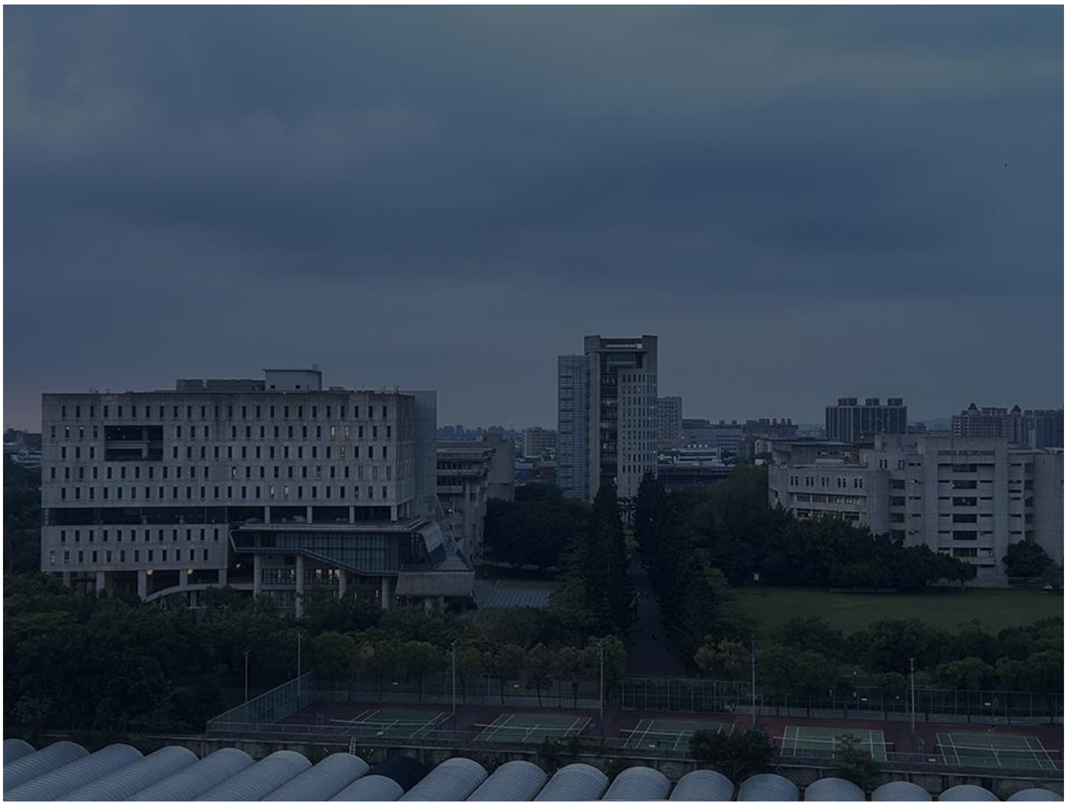
執行結果



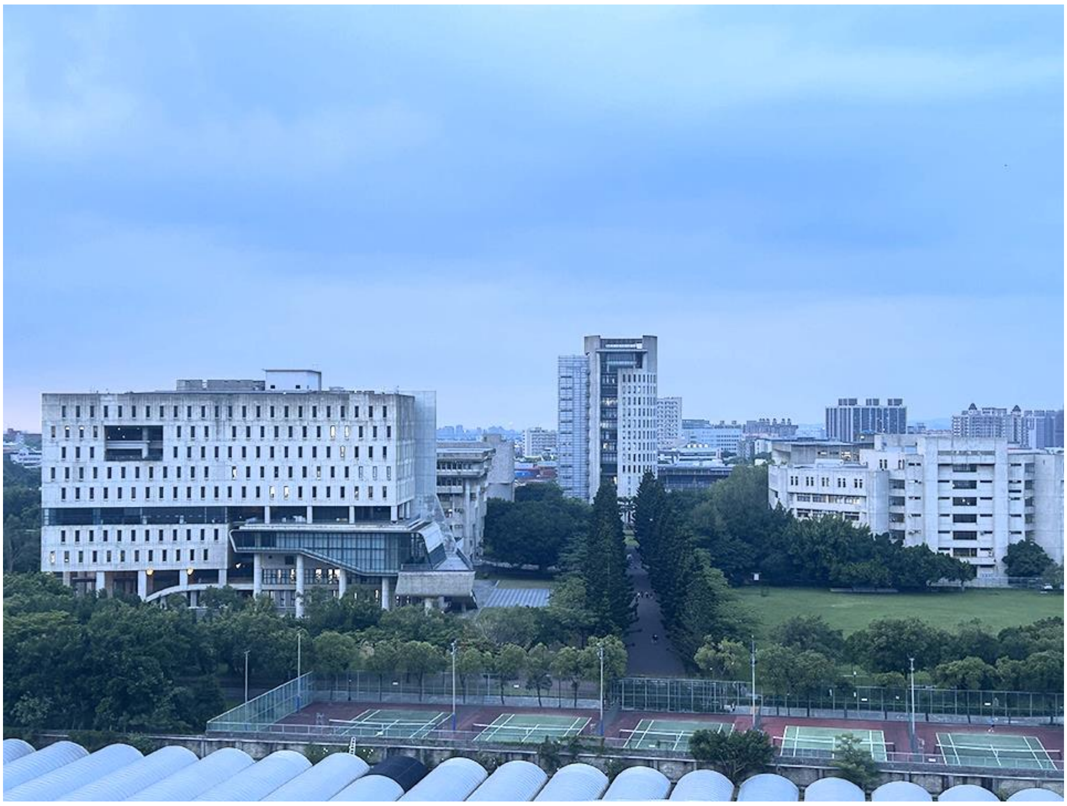
圖(一)原圖



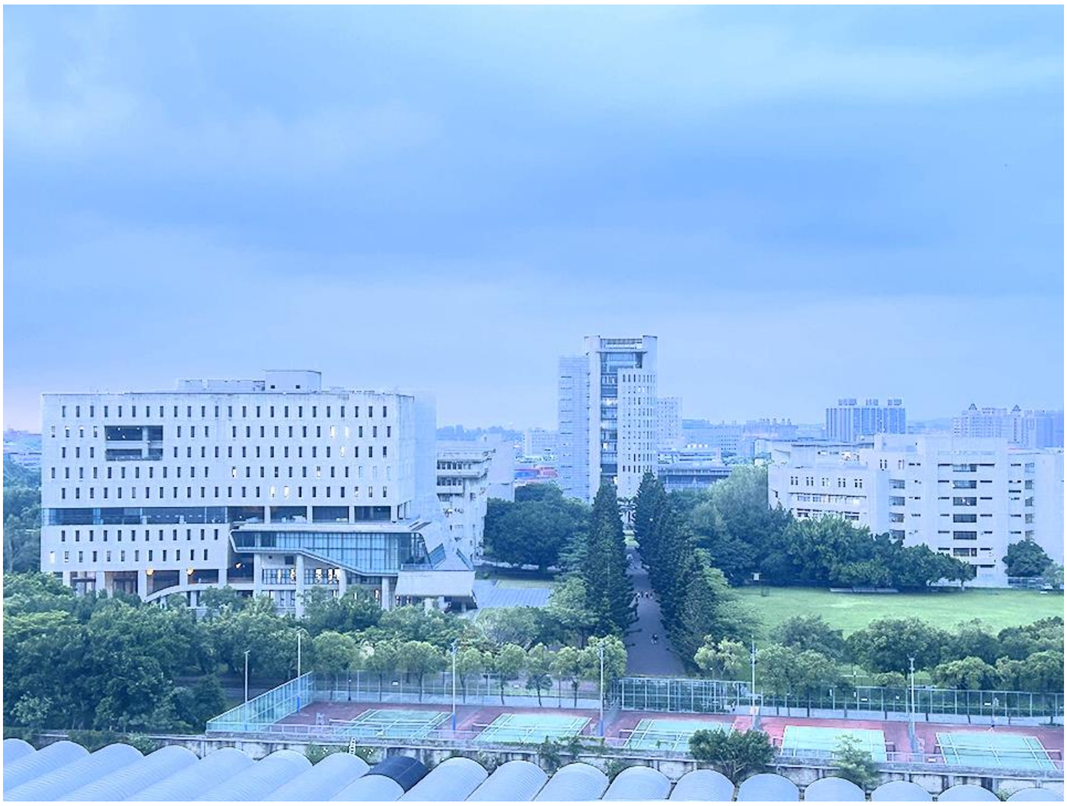
圖(二)調整數值 0.2



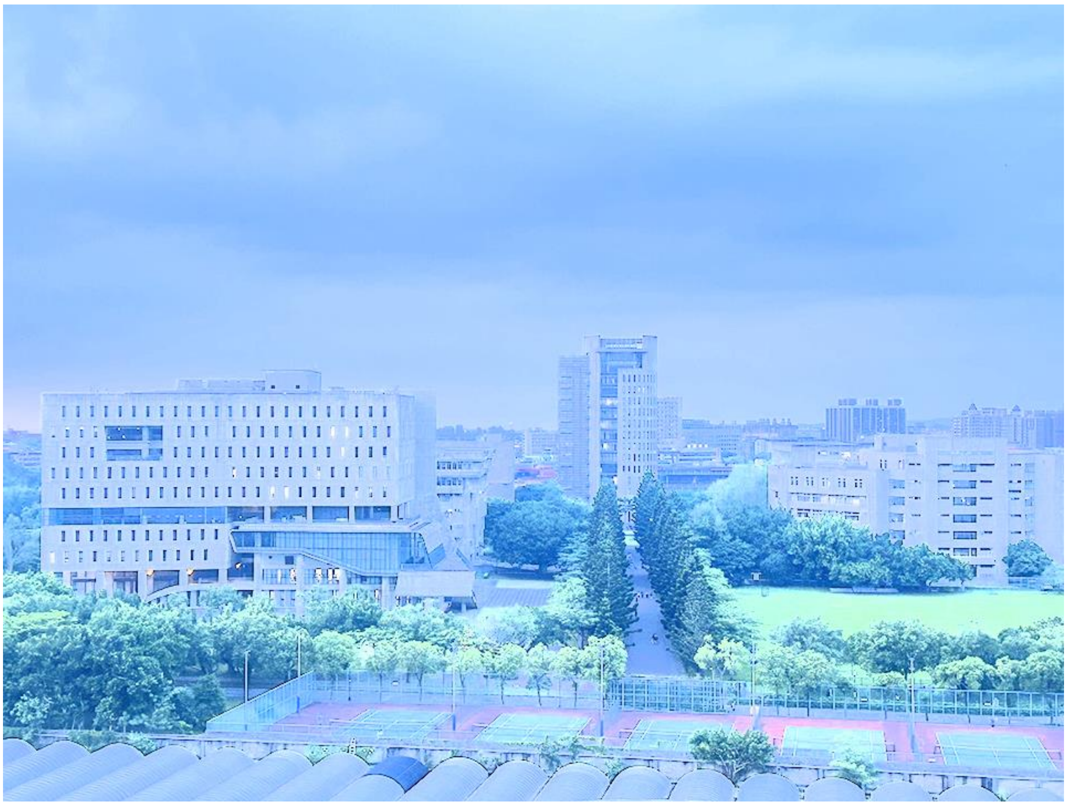
圖(三)調整數值 0.5



圖(四)調整數值 1.5



圖(五) 調整數值 2.5



圖(六) 調整數值 4

完整程式碼

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <io.h>
#include <direct.h>

using namespace std;
typedef int int32_t;
typedef short int16_t;
#pragma pack(push)
#pragma pack(1)//


class BITMAP_HEADER {
private:
    char type[2];
    int filesize;
    int reserve;
    int offset;
public:
    BITMAP_HEADER() {
        type[0] = 0;
        type[1] = 0;
        filesize = 0;
        reserve = 0;
        offset = 0;
    }
    void read(ifstream& is) {
        is.read((char*)(this), sizeof(BITMAP_HEADER));
    }
    string toString() {
        stringstream ss;
        ss << "class size: " << sizeof(BITMAP_HEADER) << endl;
        ss << "type: " << type[0] << type[1] << endl;
        ss << "file size: " << filesize << endl;
        ss << "offset: " << offset << endl;
        return ss.str();
    }
    const int FileSize() const {
        return filesize;
    }
    const int Offset() const {
        return offset;
    }
    BITMAP_HEADER& operator=(const BITMAP_HEADER& other) {
        this->type[0] = other.type[0];
        this->type[1] = other.type[1];
        this->filesize = other.filesize;
        this->reserve = other.reserve;
        this->offset = other.offset;
        return *this;
    }
};

class DIB_HEADER {
private:
    int header_size;
    int width;
    int height;
    short plane;
    short depth;
    int compress;
    int image_size;
    int resolution_horizontal;
    int resolution_vertical;
    int palette;
    int important_color;
public:
    DIB_HEADER() {
        header_size = 0;
        width = 0;
        height = 0;
        plane = 0;
        depth = 0;
        compress = 0;
        image_size = 0;
        resolution_horizontal = 0;
        resolution_vertical = 0;
        palette = 0;
        important_color = 0;
    }
    void read(ifstream& is) {
        is.read(reinterpret_cast<char*>(this), sizeof(DIB_HEADER));
    }
    string toString() {
        stringstream ss;
        ss << "header size: " << header_size << endl;
        ss << "width: " << width << endl;
        ss << "height:" << height << endl;
        ss << "plane: " << plane << endl;
        ss << "depth: " << depth << endl;
        ss << "compresion: " << compress << endl;
        ss << "image size: " << image_size << endl;
        ss << "horizontal resolution: " << resolution_horizontal <<
endl;
        ss << "vertical resolution: " << resolution_vertical << endl;
        ss << "palette: " << palette << endl;
        ss << "important color: " << important_color << endl;
        return ss.str();
    }
    int Width() {
        return width;
    }
    int Height() {
        return height;
    }
    int ImageSize() {
        return image_size;
    }
    DIB_HEADER& operator=(const DIB_HEADER& other) {
        this->header_size = other.header_size;
        this->width = other.width;
        this->height = other.height;
        this->plane = other.plane;
        this->depth = other.depth;
        this->compress = other.compress;
        this->image_size = other.image_size;
        this->resolution_horizontal = other.resolution_horizontal;
        this->resolution_vertical = other.resolution_vertical;
        this->palette = other.palette;
        this->important_color = other.important_color;
        return *this;
    }
};
class Bitmap
{
private:
    BITMAP_HEADER bitmap_header;
    DIB_HEADER dib_header;
public:
    char* buf;
    Bitmap() { buf = NULL; }
    ~Bitmap() { if (buf) delete[] buf; }
    void Load(string filename)
    {
        ifstream file;
        file.open(filename, fstream::binary);
        bitmap_header.read(file);
        dib_header.read(file);
        int rawsize = dib_header.ImageSize();
        buf = new char[rawsize];
        file.read(buf, rawsize);
        file.close();
    }
    void Write(string filename)
    {
        ofstream outfile;
        outfile.open(filename, fstream::binary);
        outfile.write((char*)(&bitmap_header),
sizeof(bitmap_header));
        outfile.write((char*)(&dib_header), sizeof(dib_header));
        int rawsize = dib_header.ImageSize();
        outfile.write(buf, rawsize);
        outfile.close();
    }
    string toString()
    {
        stringstream ss;
        ss << "[bitmap header]" << endl;
        ss << bitmap_header.toString();
        ss << "[dib header]" << endl;
        ss << dib_header.toString();
        return ss.str();
    }
    int Width()
    {
        return dib_header.Width();
    }
    int Height()
    {
        return dib_header.Height();
    }
    int imageSize()
    {
        return dib_header.ImageSize();
    }
    void Smooth()
    {
        double M[3][3] = { {1. / 9.,1. / 9., 1. / 9.} ,
                            {1. / 9.,1. / 9., 1. / 9.} ,
                            {1. / 9.,1. / 9., 1. / 9.} };
        doFilter(M);
    }
    void toGray()
    {
        int rawsize = imageSize();
        int width = Width();
        int height = Height();
        int gray = 0;
        unsigned char R = 0, G = 0, B = 0;
        int BitsPerPixel = 24;
        int RowSize = ((BitsPerPixel * width + 31) / 32) * 4;
        int i;
        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                i = y * RowSize + (x * 3) % RowSize;
                B = buf[i];
                G = buf[i + 1];
                R = buf[i + 2];
                gray = int(0.299 * R + 0.587 * G + 0.114 * B);
                if (gray > 255) gray = 255;
                if (gray < 0) gray = 0;
                buf[i] = buf[i + 1] = buf[i + 2] = gray;
            }
        }
    }
    void toBiLevel(int threshold)
    {
        int rawsize = imageSize();
        int width = Width();
        int height = Height();
        int gray = 0;
        unsigned char R = 0, G = 0, B = 0;
        int BitsPerPixel = 24;
        int RowSize = ((BitsPerPixel * width + 31) / 32) * 4;
        int i;
        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                i = y * RowSize + (x * 3) % RowSize;
                B = buf[i];
                G = buf[i + 1];
                R = buf[i + 2];
                gray = int(0.299 * R + 0.587 * G + 0.114 * B);
                if (gray > threshold) gray = 255;
                else gray = 0;
                buf[i] = buf[i + 1] = buf[i + 2] = gray;
            }
        }
    }
    void doFilter(double M[][3])
    {
        int rawsize = imageSize();
        int width = Width();
        int height = Height();
        unsigned char gray = 0;
        int BitsPerPixel = 24;
        int RowSize = ((BitsPerPixel * width + 31) / 32) * 4;
        int i;
        unsigned char** A;
        A = new unsigned char* [height];
        for (i = 0; i < height; i++)
            A[i] = new unsigned char[width];
        for (int y = 0; y < height; y++)
            for (int x = 0; x < width; x++)
            {
                i = y * RowSize + (x * 3) % RowSize;
                A[y][x] = buf[i];
            }
        int m, n;
        for (int y = 1; y < height - 1; y++)
        {
            for (int x = 1; x < width - 1; x++)
            {
                i = y * RowSize + (x * 3) % RowSize;
                gray = 0;
                for (m = 0; m < 3; m++)
                    for (n = 0; n < 3; n++)
                        gray += A[y + m - 1][x + n - 1] * M[m][n];
                buf[i] = buf[i + 1] = buf[i + 2] = gray;
```

```
                if (gray > 255) gray = 255;
                if (gray < 0) gray = 0;
            }
        }
    }

    void addBitmap(const Bitmap& other, double opacity, double
other_opcity) {
        int rawsize = imageSize();
        int width = Width();
        int height = Height();
        unsigned char gray = 0;
        unsigned char R = 0, G = 0, B = 0;
        unsigned char other_R = 0, other_G = 0, other_B = 0;
        int BitsPerPixel = 24;
        int RowSize = ((BitsPerPixel * width + 31) / 32) * 4;
        int i;
        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                i = y * RowSize + (x * 3) % RowSize;
                B = buf[i];
                G = buf[i + 1];
                R = buf[i + 2];
                double r, g, b, other_r, other_g, other_b;
                r = static_cast<double>(R);
                g = static_cast<double>(G);
                b = static_cast<double>(B);
                other_B = other.buf[i];
                other_G = other.buf[i + 1];
                other_R = other.buf[i + 2];
                other_r = static_cast<double>(other_R);
                other_g = static_cast<double>(other_G);
                other_b = static_cast<double>(other_B);
                R = r * opacity + other_r * other_opcity;
                G = g * opacity + other_g * other_opcity;
                B = b * opacity + other_b * other_opcity;
                buf[i] = B;
                buf[i + 1] = G;
                buf[i + 2] = R;
            }
        }
    }

    /*****************************************************************
        Value need to be a float num greater than zero,
        0~1 is decrease brightness, 1~2 is increase brightness
        if Value is more bigger, the result will be brighter but it
        maybe isn't a good result.
    *****************************************************************/
    void Brightness(double Value) {
        int rawsize = imageSize();
        int width = Width();
        int height = Height();
        int BitsPerPixel = 24;
        int RowSize = ((BitsPerPixel * width + 31) / 32) * 4;
        int i;

        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                i = y * RowSize + (x * 3) % RowSize;
                // Read RGB Value
                unsigned char b = buf[i];
                unsigned char g = buf[i + 1];
                unsigned char r = buf[i + 2];

                // Normalization let RGB's value between 0~1
                double R = static_cast<double>(r) / 255;
                double G = static_cast<double>(g) / 255;
                double B = static_cast<double>(b) / 255;

                // RGB To HSV Formula
                double H, S, V;
                double Max = max(max(R, G), B);
                double Min = min(min(R, G), B);
                double Delta = Max - Min;

                if (Delta == 0) { // Define H
                    H = 0;
                }
                else if (Max == R) {
                    H = 60 * (fmod(((G - B) / Delta), 6));
                }
                else if (Max == G) {
                    H = 60 * ((B - R) / Delta + 2);
                }
                else if (Max == B) {
                    H = 60 * ((R - G) / Delta + 4);
                }
                if (H < 0) {
                    H += 360;
                }

                if (Max == 0) { // Define S
                    S = 0;
                }
                else {
                    S = Delta / Max;
                }
                V = Max; // Define V
                // END of RGB To HSV Formula

                // Adjust V's Value
                V = max(0.0,min(V * Value, 1.0));


                // HSV To RGB Formula
                double C = V * S;
                double h = fmod(H / 60, 6);
                double X = C * (1 - abs(fmod(h, 2) - 1));
                double m = V - C;
```

```
r = g = b = 0;

if (0 <= h && h < 1) {
    r = (C + m) * 255;
    g = (X + m) * 255;
    b = (m) * 255;
}
else if (1 <= h && h < 2) {
    r = (X + m) * 255;
    g = (C + m) * 255;
    b = (m) * 255;
}
else if (2 <= h && h < 3) {
    r = (m) * 255;
    g = (C + m) * 255;
    b = (X + m) * 255;
}
else if (3 <= h && h < 4) {
    r = (m) * 255;
    g = (X + m) * 255;
    b = (C + m) * 255;
}
else if (4 <= h && h < 5) {
    r = (X + m) * 255;
    g = (m) * 255;
    b = (C + m) * 255;
}
else if (5 <= h && h < 6) {
    r = (C + m) * 255;
    g = (m) * 255;
    b = (X + m) * 255;
}
// END of HSV To RGB Formula

// Store RGB to buf[]
buf[i] = b;
buf[i + 1] = g;
buf[i + 2] = r;
```

```
            }
        }
    }

    Bitmap& operator= (const Bitmap& other) {
        this->bitmap_header = other.bitmap_header;
        this->dib_header = other.dib_header;
        int rawsize = imageSize();
        if (this->buf == NULL) {
            this->buf = new char[rawsize];
        }
        else {
            delete[] this->buf;
            this->buf = new char[rawsize];
        }
        for (int i = 0; i < rawsize; i++) {
            this->buf[i] = other.buf[i];
        }
        return *this;
    }
};
#pragma pack(pop)
void make_dir(const string dir_name)
{
    if (_access(dir_name.c_str(), 0) == -1)
        int re = _mkdir(dir_name.c_str());
}
void BitmapTest() {
    double M2[3][3] = { {1,0,-1} ,{2,0,-2} ,{1,0,-1} }; //綜向
    double M3[3][3] = { {1,2,1} ,{0,0,0} ,{-1,-2,-1} }; //橫向
    Bitmap bmp, bmp2, bmp3, bmp4, bmp5;

    string path = "BitmapTest";
    make_dir(path);

    bmp.Load("campus.bmp");
    bmp2 = bmp;
    bmp.Brightness(2);
    bmp.Write(path+"/result1.bmp");
    bmp2.toGray();
    bmp2.Write(path + "/result2.bmp");
    bmp2.Smooth();
    bmp2.Write(path + "/result3.bmp");
    bmp2.toBiLevel(100);
    bmp2.Write(path + "/result4.bmp");
    bmp2.doFilter(M2);
    bmp2.Write(path + "/result5.bmp");
    bmp2.addBitmap(bmp, 0.5, 0.5);
    bmp2.Write(path + "/result6.bmp");
}
void BrightTest() {
    Bitmap bmp, bmp2, bmp3, bmp4, bmp5;

    string path = "BrightTest";
    make_dir(path);

    bmp.Load("campus.bmp");
    bmp2 = bmp3 = bmp4 = bmp5 = bmp;

    bmp.Brightness(0.2);
    bmp.Write(path + "/result1.bmp");

    bmp2.Brightness(0.5);
    bmp2.Write(path + "/result2.bmp");

    bmp3.Brightness(1.5);
    bmp3.Write(path + "/result3.bmp");

    bmp4.Brightness(2.5);
    bmp4.Write(path + "/result4.bmp");

    bmp5.Brightness(4);
    bmp5.Write(path + "/result5.bmp");
}
```

```
int main(int argc, char** argv) {
    BrightTest();
    BitmapTest();
    return 0;
}
```