

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



LÊ QUANG MINH – 20120329

BÁO CÁO ĐỒ ÁN
XỬ LÝ ẢNH BMP BẰNG C/C++

| Giáo viên hướng dẫn |
ThS. Phạm Minh Hoàng

Khoa Công Nghệ Thông Tin
Thành phố Hồ Chí Minh – 2021

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



LÊ QUANG MINH – 20120329

BÁO CÁO ĐỒ ÁN
XỬ LÝ ẢNH BMP BẰNG C/C++

| Giáo viên hướng dẫn |
ThS. Phạm Minh Hoàng

Khoa Công Nghệ Thông Tin
Thành phố Hồ Chí Minh – 2021

LỜI CẢM ƠN

Em xin cảm ơn ThS. Phạm Minh Hoàng đã cho tụi em cơ hội để được thực hiện đồ án xử lý hình ảnh này để hiểu hơn về tập tin nhị phân và cách thức hoạt động của một file ảnh BMP.

Các comment trong các dòng code em sẽ viết bằng Tiếng Anh (mặc dù có một chỗ viết Tiếng Việt) để làm quen với công việc làm sau này.

MỤC LỤC

MỤC LỤC.....	4
DANH MỤC HÌNH	5
DANH MỤC BẢNG	6
DANH MỤC TỪ VIẾT TẮT.....	7
I. GIỚI THIỆU ĐỒ ÁN.....	8
II. HEADER CỦA CHƯƠNG TRÌNH.....	9
III. NHẬP ẢNH TỪ TRONG Ổ ĐĨA.....	11
IV. HÀM XUẤT ẢNH RA Ổ ĐĨA.....	12
V. HÀM CHUYỂN ẢNH 24BPP VỀ 8BPP	13
VI. HÀM CHUYỂN ẢNH 32BPP VỀ 8BPP	15
VII. HÀM RESCALE ẢNH 8BPP	17
VIII. HÀM RESCALE ẢNH 24BPP	20
IX. HÀM RESCALE ẢNH 32BPP	22
X. HÀM MAIN.....	24

DANH MỤC HÌNH

DANH MỤC BẢNG

DANH MỤC TỪ VIẾT TẮT

I. GIỚI THIỆU ĐỒ ÁN

Đây là đồ án của môn Kỹ Thuật Lập Trình, ý tưởng rằng chúng ta sử dụng các thao tác với file nhị phân trong C/C++ để có thể thực hiện các tiến trình xử lý ảnh mà chúng ta mong muốn.

Các đề mục trong bài đồ án sẽ được kèm theo code C/C++ và được chia thành từng phần chính:

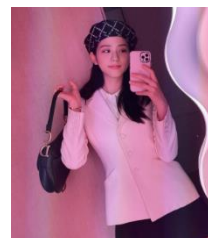
- Nhập ảnh BMP từ một address/directory trên máy tính vào trong chương trình.
- Xuất một ảnh BMP từ chương trình ra thành một file trên máy tính.
- Chuyển đổi một ảnh BMP trở về dạng trắng đen (24/32bpp về 8bpp)
- Thu nhỏ một ảnh BMP theo một tỉ lệ S cho trước

Một số demo của từng phần:

- Convert ảnh về 8bpp



- Thu nhỏ ảnh (Ảnh dưới đây được thu nhỏ đi 4 lần)



II. HEADER CỦA CHƯƠNG TRÌNH

Chúng ta phải tách file ở mỗi chương trình để cho code thuận tiện và việc code trở nên chuyên nghiệp hơn.

```
#pragma once
#pragma pack(2)
#include <iostream>
#include <fstream>
using namespace std;

struct Header { // 14 bytes
    unsigned char signature[2];
    unsigned int FileSize;
    unsigned int reserved;
    unsigned int DataOffset; // contain the address of the point begin to save the picture
};

struct DIB { // 40 bytes
    unsigned int DIBsize;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bpp; // bits per pixel
    unsigned int compression;
    unsigned int imageSize;
    unsigned int Xpixels;
    unsigned int Ypixels;
    unsigned int colorsUsed;
    unsigned int importantColor;
};

struct bmp { // one picture will have a struct like this
    Header header;
    DIB dib;
    char* colorpalette;
    char* data; // contain the pixel data
};

struct Pixel_24bit {
    unsigned char Blue; // Thuc su la em da thu moi cach la de kieu du lieu la char,int,unsigned int
    // nhưng ma nó ko có ra được cái màu đẹp như đề unsigned char
    unsigned char Green;
    unsigned char Red;
};

struct Pixel_32bit {
    unsigned char A;
    unsigned char Blue;
    unsigned char Green;
    unsigned char Red;
};

int readBMP(bmp* &picture, const char* filename); // put the file bmp into a bmp structure
void writeBMP(bmp* picture, const char* filename);
void rescale_24bit(bmp* &picture, const char* output, int s);
void rescale_32bit(bmp* &picture, const char* output, int s);
void grayscale_24bit(bmp* &picture, const char* output);
void writeBMP_8bit(bmp* picture, const char* filename);
void grayscale_32bit(bmp* &picture, const char* output);
void rescale_8bit(bmp* &picture, const char* output, int s);
```

Giải thích về Header :

- Việc sử dụng `#pragma pack(2)`, trong quá trình code, em có sử dụng biến short chiếm 2 bytes trong bộ nhớ. Vì lý do trên, bộ nhớ sẽ chèn 2 bytes thừa vào giữa biến short để biến nó thành 4 bytes, vì vậy lúc em khai `sizeof(NameOfStruct.DIB)` sẽ không chính xác là 40 byte.
- Ở struct `Pixel_24bit` và `Pixel_32bit`, lý do em tách nó ra thành hai struct rằng để tiết kiệm trong quá trình chúng ta không có byte thừa trên bộ nhớ (có một dòng if để kiểm tra rằng ảnh là 24bpp hay 32bpp để cấp phát chính xác), và việc sử dụng riêng lẻ như vậy cũng khiến việc tách hàm mỗi câu thành 2 loại xử lý riêng cho ảnh 24bpp và 32bpp trở nên dễ hơn.
- Ở trong struct pixel, em có ghi rằng việc sử dụng biến kiểu unsigned char sẽ cho ra màu đẹp và chính xác hơn, ở đây em đã thử sử dụng int, unsigned int và char nhưng đều cho ra màu khá là không đúng, có thể rằng việc giới hạn giá trị của unsigned char sẽ cho ra giá trị trong khoảng [0,255] đã giải quyết điều này.

III. NHẬP ẢNH TỪ TRONG Ổ ĐĨA

```
void readBMP(bmp*& picture, const char* filename) // put the file bmp into a bmp structure
{
    fstream input;
    input.open(filename, ios::in | ios::binary); // the filename should be ended with .bmp
    if (!input.is_open())
    {
        cout << "The image is not loaded successfully";
        return;
    }
    else
    {
        cout << "The image is loaded successfully";
    }
    picture = new bmp;
    input.read((char*)&picture->header, 14);
    input.read((char*)&picture->dib, 40);
    picture->colorpalette = new char[picture->header.DataOffset - 54];


    //Bản chất là đọc phần còn lại của DIB
    input.read(picture->colorpalette, picture->header.DataOffset - 54);
    //*****

    input.seekg(picture->header.DataOffset, ios::beg); // move the pointer to the DataOffset's position
    unsigned int w, h, size;
    w = picture->dib.width;
    h = picture->dib.height;
    unsigned int BitPerPixel = (picture->dib.bpp) / 8; // determine whether the bmp file is 8pp, 24pp
    or 32pp
    unsigned int padding = (4 - (w % 4)) % 4;
    size = w * h * BitPerPixel + padding * h;
    picture->data = new char[size];
    input.read(picture->data, size);
    input.close();
}
```

Giải thích về hàm nhập ảnh:

- Bản chất rằng em sẽ cấp phát động một file ảnh bmp ở trong hàm MAIN và bỏ vào hàm này ở tham chiếu => kết thúc hàm ta sẽ có được một con trỏ kiểu bmp và toàn bộ ảnh sẽ được lưu ở Heap.
- Ở khúc đọc picture->colorpalette, mục đích thật sự của em không phải là em đọc hết color table vì ảnh 24/32 nhiều ảnh không có color table. Mục đích của em là đọc phần thừa của DIB vì trong quá trình thực hiện vì có nhiều ảnh không thể đọc được ở trình đọc ảnh của Windows nếu chúng ta không đọc hết phần thừa đó.

IV. HÀM XUẤT ẢNH RA Ổ ĐĨA



```
void writeBMP(bmp* picture, const char* filename)
{
    fstream output;
    output.open(filename, ios::out | ios::binary);
    output.write((char*)&picture->header, sizeof(Header));
    output.write((char*)&picture->dib, sizeof(DIB));

    //Bản chat that ra là vietphan con thua của DIB
    output.write(picture->colorpalette, picture->header.DataOffset - 54);

    // Print out the picture file
    output.seekp(picture->header.DataOffset, ios::beg); // Move the pointer to the place starting to
begin the data pixel
    output.write(picture->data, picture->header.FileSize);
    output.close();
}
```

Giải thích về hàm xuất ảnh:

- Ở khúc viết các byte trong data pixel, em cho load hết số byte bằng FileSize vì để chắc chắn rằng sẽ đọc được hết các byte đã được ghi trong mảng picture->data (Tránh hạn chế sai sót).

V. HÀM CHUYỂN ẢNH 24BPP VỀ 8BPP

```

void grayscale_24bit(bmp* & picture, const char* output)
{
    bmp* temp = new bmp; // remember to delete
    *temp = *picture;

    // Get the data pixel size
    unsigned int w, h, size;
    w = picture->dib.width;
    h = picture->dib.height;
    unsigned int BitPerPixel = (picture->dib.bpp) / 8;
    unsigned int padding = (4 - (w * BitPerPixel % 4)) % 4;
    size = w * h * BitPerPixel + padding * h;

    //*****
    //Create a temporary pointer to go through every element of the pixeldata array
    char* temp_ = temp->data;
    //*****

    //Create a Pixel-Image Array ( 24Bit ) ( this will be a 2D Array for easier manipulation )
    Pixel_24bit* image = new Pixel_24bit [w*h];
    //*****

    //Get the Data of 2D Pixel-Image
    int count = 0;
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            image[count].Blue = *(temp_++);
            image[count].Green = *(temp_++);
            image[count].Red = *(temp_++);
            count++;
        }
        for (int k = 0; k < padding; k++)
        {
            *(temp_++);
        }
    }

    //Create the new Dynamically Allocated GrayScaleArray
    int padding_new = (4 - (w % 4)) % 4;
    int size_new = w * h + h * padding_new;
    temp->data = new char[size_new];
    count = 0;
    temp_ = temp->data;

    //gray-scaling process
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            //Calculating the average value
            int Average = image[count].Blue / 3 + image[count].Green / 3 + image[count].Red / 3;
            image[count].Blue = image[count].Green = image[count].Red = Average;
            //Put the value into the pixel data
            *(temp_++) = Average;
            //increase the index
            count++;
        }
        for (int j = 0; j < padding_new; j++)
        {
            *(temp_++) = 0;
        }
    }

    //write 1024 color of the colormap
    temp->colorpalette = new char[256 * 4];
    temp_ = temp->colorpalette; // reuse the temp_ pointer
    count = 0;
    for (int i = 0; i <= 255; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            *(temp_++) = i;
        }
        *(temp_++) = 0;
    }

    temp->header.FileSize = 54 + 1024 + size_new;
    temp->dib.bpp = 8;
    temp->dib.OIBSize = 40;
    temp->dib.ImageSize = size_new;
    temp->header.DataOffset = 54 + 1024;
    writeBMP(temp, output);

    //delete Dynamically allocated array
    delete[] image;
    delete[] temp->data;
    delete[] temp->colorpalette;
    delete temp;
}

```

Giải thích hàm chuyển 24bpp về 8bpp:

- Ở đây em muốn tách ra thành hai hàm xử lý riêng biệt cho 24bpp và 32bpp (Để dễ kiểm soát hơn trong quá trình code, em muốn biết được lỗi sai ở đâu trong quá trình làm để dễ debug, nếu ta đưa 2 trường hợp về cùng một hàm thì khá là phức tạp trong quá trình xử lý)
- Ý tưởng của em là sẽ cấp phát động một cấu trúc bmp mới (đặt tên là temp) kèm với một con trỏ temp_ để có thể linh hoạt đi qua từng vị trí các mảng động (để không bị mất vị trí con trỏ ban đầu)
- Ở phần chuyển này em sử dụng mảng một chiều của cấu trúc Pixel_24bit vì đối với hàm này thì thao tác trên mảng một chiều sẽ dễ dàng hơn.
- Về ảnh 8bpp, nó sẽ không được đọc nếu thiếu color table, vì vậy em đã đi một vòng lặp để viết 1024 byte cho color table của ảnh.
- Hàm này có thể đọc được ảnh đã được rescale (thực hiện ở câu 4, mục VIII và IX) để biến ảnh về ảnh 8bpp.

Lưu ý: Các hàm ở mục VI và VII không thể được thực hiện với ảnh 8bpp vì đi ngược mục đích chuyển ảnh 24/32bpp về 8bpp, sẽ có một thông báo nếu bạn thực hiện nhập ảnh 8bpp vào và chọn convert8bit

VI. HÀM CHUYỂN ẢNH 32BPP VỀ 8BPP

```

void grayscale_32bit(bmp* picture, const char* output)
{
    bmp* temp = new bmp; // remember to delete
    *temp = *picture;

    // Get the data pixel size
    unsigned int w, h, size;
    w = picture->dib.width;
    h = picture->dib.height;
    unsigned int BitPerPixel = (picture->dib.bpp) / 8;
    unsigned int padding = (4 - (w * BitPerPixel % 4)) % 4;
    size = w * h * BitPerPixel + padding * h;

    //*****
    //Create a temporary pointer to go through every element of the pixeldata array
    char* temp_ = temp->data;
    //*****

    //Create a Pixel-Image Array ( 24Bit ) ( this will be a 2D Array for easier manipulation )
    Pixel_32bit** image = new Pixel_32bit * [h];
    for (int i = 0; i < h; i++)
    {
        image[i] = new Pixel_32bit[w];
    }
    //*****

    //Get the Data of 2D Pixel-Image
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            image[i][j].A = *(temp_++);
            image[i][j].Blue = *(temp_++);
            image[i][j].Green = *(temp_++);
            image[i][j].Red = *(temp_++);
        }
    }

    //Create the new Dynamically Allocated GrayScaleArray
    int padding_new = (4 - (w % 4)) % 4;
    int size_new = w * h + h * padding_new;
    temp->data = new char[size_new];
    int count = 0;

    //gray-scaling process
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            int Average = image[i][j].Blue / 4 + image[i][j].Green / 4 + image[i][j].Red / 4 + image[i][j].A / 4;
            image[i][j].Blue = image[i][j].Green = image[i][j].Red = Average;
        }
    }

    //put all the pixel into array and transform into 8bit
    temp_ = temp->data;
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            *(temp_++) = image[i][j].Red; // just take a random blue red or green or A
        }
        for (int k = 0; k < padding_new; k++)
        {
            *(temp_++) = 0;
        }
    }

    //write 1024 color of the colortable
    temp->colorpalette = new char[256 * 4];
    temp_ = temp->colorpalette; // reuse the temp_ pointer
    count = 0;
    for (int i = 0; i < 255; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            *(temp_++) = i;
        }
        *(temp_++) = 0;
    }

    temp->header.FileSize = 54 + 1024 + size_new;
    temp->dib.bpp = 8;
    temp->dib.DIBSize = 40;
    temp->dib.imageSize = size_new;
    temp->header.DataOffset = 54 + 1024;
    writeBMP(temp, output);

    //delete Dynamically allocated array
    for (int i = 0; i < h; i++)
    {
        delete[] image[i];
    }
    delete[] image;
    delete[] temp->data;
    delete[] temp->colorpalette;
    delete temp;
}

```

Giải thích hàm:

- Mặc dù ở phần chuyển từ 24bpp về 8bpp em sử dụng mảng một chiều, nhưng ở phần này em muốn thử nghiệm rằng việc sử dụng mảng hai chiều sử dụng hai con trỏ (một con trỏ cấp 1 và một con trỏ cấp 2) liệu có thực thi hay không.
- Hàm này cơ bản chỉ khác ở hàm chuyển đổi 24bpp ở chỗ mảng hai chiều, việc tính toán lại các giá trị cho DIB cũng tương tự như vậy.
- Tương tự mục VI, hàm này có thể đọc được ảnh đã được rescale (thực hiện ở câu 4, mục VII, VIII và IX) để biến ảnh về ảnh 8bpp.

Lưu ý: Các hàm ở mục V và VI không thể được thực hiện với ảnh 8bpp vì đi ngược mục đích chuyển ảnh 24/32bpp về 8bpp, sẽ có một thông báo nếu bạn thực hiện nhập ảnh 8bpp vô và chọn convert8bit.

VII. HÀM RESCALE ẢNH 8BPP

Ở đề mục này em thiết kế 2 hàm, một hàm chính để xử lý các thông số (đồng thời đi 2 vòng lặp duyệt mảng 2 chiều để gọi xử lý thuật toán) và một hàm đi các thuật toán xử lý riêng, em sẽ trình bày hàm đi thuật toán xử lý và vòng lặp gọi hàm đó trước.

```
//Calculating the new size of the new image
unsigned int h_new, w_new, padding_new;
if (h % s == 0)
{
    h_new = h / s;
}
else
{
    h_new = h / s + 1;
}
if (w % s == 0)
{
    w_new = w / s;
}
else w_new = w / s + 1;
padding_new = (4-(w_new % 4))%4;
int size_new = w_new * h_new + padding_new * h_new; // 8bit

//Creating the rescale image pixel data 2D Array
char* image_new = new char[w_new * h_new];
//*****

int count = 0;
for (int i = 0; i < h; i += s)
{
    for (int j = 0; j < w; j += s)
    {
        if (i == (h - (h % s)) && h / s != 0) // h/s !=0 in order to not get the error from c/0
        {
            image_new[count++] = countAverage8bit(image, i, j, h % s, s); // leftover byte
            continue;
        }
        if (j == (w - (w % s)) && w / s != 0)
        {
            image_new[count++] = countAverage8bit(image, i, j, s, w % s); // leftover byte
            continue;
        }
        image_new[count++] = countAverage8bit(image, i, j, s, s);
    }
}
```

```
char countAverage8bit(char** a, int i_cur, int j_cur, int i_increase, int j_increase)
{
    int divide = i_increase * j_increase;
    int temp;
    int sum = 0;
    for (int i = i_cur; i < i_cur + i_increase; i++)
    {
        for (int j = j_cur; j < j_cur + j_increase; j++)
        {
            sum += a[i][j];
        }
    }
    temp = sum / divide;
    return temp;
}
```

Giải thích hàm:

- Ở đây em sẽ coi pixel data như là một mảng hai chiều, vì thế em truyền một con trỏ cấp 2 vô chứa các data pixel (8bpp một pixel chỉ là một byte char)
- Ý tưởng rằng em sẽ đi 2 vòng lặp duyệt mảng 2 chiều, bước nhảy cho i và j là S (tỉ lệ giảm), tại mỗi vị trí khi vòng lặp đi tới, ta sẽ đi một sub-array SxS, tính trung bình cộng lại và bỏ giá trị đó vô một mảng một chiều image_new[] mới, sau đó sử dụng cái paddingbyte, width, height đã được tính lại, bỏ lại vô biến bmp temp và thực hiện như phần cuối của câu 3 (Mục VI, VII).
- Và trong quá trình đi vòng lặp, sẽ có những lúc ta không thể đi được sub-array SxS tại vị trí đó (những vị trí thừa ở góc cuối phải và góc trái dưới mảng 2 chiều), vì vậy em có đặt một dòng if để thực hiện việc đó (tính trung bình cộng của các ô thừa trong đó và chắc chắn số ô thừa < SxS).
- Ở hàm countAverage8bit, tham số đầu vào của em sẽ là một con trỏ trỏ tới mảng 2 chiều, int i_cur và int j_cur chính là để nhận vị trí hiện tại của ô pixel ở vòng lặp trên. Điều đặc biệt trong hàm này chính là hai biến tham trị int i_increase và int j_increase. Hai biến này chính là để biểu thị rằng sub-array đi sẽ là như thế nào. Nếu như bình thường thì cả hai biến trên sẽ là bằng S => Đi một ma trận con SxS thì ở những vị trí đi byte thừa như đã nhắc ở trên, chỉ việc thay i_increase hoặc j_increase bằng giá trị thừa của quá trình đi dòng hoặc cột thì ta có thể dễ dàng tính được giá trị trung bình mà không cần viết một hàm đặc biệt riêng cho khúc đó. Tận dụng được một hàm để đi hết mọi trường hợp. Hàm này trả về một char để thêm vào phần tử mảng 2 chiều chứa các datapixel mới.
- Tất cả các hàm dành cho 23/32bpp còn lại sẽ bám theo quy luật của hàm 8bpp này.

Code đầy đủ của hàm scale 8bpp

```

char countAverage8bit(char* a, int i_cur, int j_cur, int i_increas, int j_increas)
{
    int divide = i_increas * j_increas;
    int temp;
    int sum = 0;
    for (int i = i_cur; i < i_cur + i_increas; i++)
    {
        for (int j = j_cur; j < j_cur + j_increas; j++)
        {
            sum += a[i][j];
        }
    }
    temp = sum / divide;
    return temp;
}

void rescale_8bit(bmp* picture, const char* output, int s)
{
    bmp* temp = new bmp; // remember to delete
    *temp = *picture;

    // Get the data pixel size
    unsigned int w, h, size;
    w = picture->dib.width;
    h = picture->dib.height;
    unsigned int BitPerPixel = (picture->dib.bpp) / 8;
    unsigned int padding = w * BitPerPixel % 4;
    size = w * h * BitPerPixel + padding * h;

    //*****
    //Create a temporary pointer to go through every element of the pixeldata array
    char* temp_data = temp->data;
    //*****

    //Create a Pixel Image Array ( 24bit ) ( this will be a 2D Array for easier manipulation )
    char* image = new char* [h];
    for (int i = 0; i < h; i++)
    {
        image[i] = new char[w];
    }
    //*****

    //Get the Data of 2D-Pixel Image
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            image[i][j] = *(temp_data++);
        }
        for (int k = 0; k < padding; k++)
        {
            *(temp_data++) = 0;
        }
    }

    //Calculating the new size of the new image
    unsigned int h_new, w_new, padding_new;
    if (h % s == 0)
    {
        h_new = h / s;
    }
    else
    {
        h_new = h / s + 1;
    }
    if (w % s == 0)
    {
        w_new = w / s;
    }
    else
    {
        w_new = w / s + 1;
    }
    padding_new = ((w_new % 4) + 4) % 4;
    int size_new = w_new * h_new + padding_new * h_new; // 8bit

    //Creating the rescaled image pixel data 2D Array
    char* image_new = new char[w_new * h_new];
    //*****

    int count = 0;
    for (int i = 0; i < h; i += s)
    {
        for (int j = 0; j < w; j += s)
        {
            if (i == (h - (h % s)) && h / s != 0) // h/s != 0 is order to not get the error from c/d
            {
                image_new[count++] = countAverage8bit(image, i, j, h % s, s); // leftover byte
                continue;
            }
            if (j == (w - (w % s)) && w / s != 0)
            {
                image_new[count++] = countAverage8bit(image, i, j, s, w % s); // leftover byte
                continue;
            }
            image_new[count++] = countAverage8bit(image, i, j, s, s);
        }
    }

    //Put all the value into the temp-data
    temp_data = new char[size_new];
    temp_data = temp->data;
    count = 0;
    for (int i = 0; i < h_new; i++)
    {
        for (int j = 0; j < w_new; j++)
        {
            *(temp_data++) = image_new[count++];
        }
        for (int k = 0; k < padding_new; k++)
        {
            *(temp_data++) = 0;
        }
    }

    //calculate new Header and DIB
    temp->dib.height = h_new;
    temp->dib.width = w_new;
    temp->header.FileSize = 54 + 1074 + size_new;
    temp->dib.ImageSize = size_new;
    writeBMP(temp, output);

    //delete dynamic array
    delete[] image_new;
    for (int i = 0; i < h; i++)
    {
        delete[] image[i];
    }
    delete image;
    delete[] temp_data;
    delete temp;
}

```

VIII. HÀM RESCALE ẢNH 24BPP

```

Pixel_24bit countAverage24bit(Pixel_24bit* a, int l_cur, int l_incre, int l_incre, int l_incre)
{
    unsigned char divide = unsigned char((l_incre) * unsigned char(j_incre));
    Pixel_24bit temp;
    int Sum_Blue = 0, Sum_Green = 0, Sum_Red = 0;
    for (int i = l_cur; i < l_cur + l_incre; i++)
    {
        for (int j = j_cur; j < j_cur + j_incre; j++)
        {
            Sum_Blue += a[i][j].Blue;
            Sum_Green += a[i][j].Green;
            Sum_Red += a[i][j].Red;
        }
    }
    temp.Blue = Sum_Blue / divide;
    temp.Green = Sum_Green / divide;
    temp.Red = Sum_Red / divide;
    return temp;
}

void rescale_24bit(Picture* picture, const char* output, int s) // remember to add output directory for the 5th question
{
    bmp* temp = new bmp; // remember to delete
    *temp = *picture;

    // Get the data pixel size
    unsigned int w, h, stride;
    w = picture->data->width;
    h = picture->data->height;
    unsigned int BitPerPixel = (picture->data->bpp) / 8;
    unsigned int padding = (4 - (w * BitPerPixel % 4)) * 4;
    size = w * h * BitPerPixel + padding * h;

    //Create a temporary pointer to go through every element of the pixeldata array
    char* temp_data;
    temp_data = new char[size];

    //Create a Pixel Image Array (24bit) (this will be a 2D Array for easier manipulation)
    Pixel_24bit** image = new Pixel_24bit*[h];
    for (int i = 0; i < h; i++)
    {
        image[i] = new Pixel_24bit[w];
    }

    //Get the data of 24 Pixel-Image
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            image[i][j].Blue = *(temp_data + i * w * 3 + j * 3);
            image[i][j].Green = *(temp_data + i * w * 3 + j * 3 + 1);
            image[i][j].Red = *(temp_data + i * w * 3 + j * 3 + 2);
        }
    }

    //Calculating the new size of the new image
    unsigned int h_new, w_new, padding_new;
    if (h % s == 0)
    {
        h_new = h / s;
    }
    else
    {
        h_new = h / s + 1;
    }
    if (w % s == 0)
    {
        w_new = w / s;
    }
    else
    {
        w_new = w / s + 1;
    }
    padding_new = (4 - (w_new * 3 % 4)) * 4;
    int size_new = w_new * h_new * 3 + padding_new * h_new; // 24bit

    //Creating the rescale image pixel data 2D Array
    Pixel_24bit** image_new = new Pixel_24bit*[h_new];
    for (int i = 0; i < h_new; i++)
    {
        image_new[i] = new Pixel_24bit[w_new];
    }

    int count = 0;
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            if ((i - (h % s)) % s == 0) // h/s - 1 in order to not get the error from s/0
            {
                image_new[count++] = countAverage24bit(image, i, j, h % s, s);
                continue;
            }
            if ((j - (w % s)) % s == 0)
            {
                image_new[count++] = countAverage24bit(image, i, j, s, w % s);
                continue;
            }
            image_new[count++] = countAverage24bit(image, i, j, s, s);
        }
    }

    //Put all the value into the temp_data
    temp_data = new char[size_new];
    temp_data = temp_data;
    count = 0;
    for (int i = 0; i < h_new; i++)
    {
        for (int j = 0; j < w_new; j++)
        {
            *(temp_data + count++) = image_new[i][j].Blue;
            *(temp_data + count++) = image_new[i][j].Green;
            *(temp_data + count++) = image_new[i][j].Red;
            count++;
        }
    }
    for (int k = 0; k < padding_new; k++)
    {
        *(temp_data + count++) = 0;
    }
    temp->data->height = h_new;
    temp->data->width = w_new;
    temp->header->FileSize = 14 + picture->data->DIBsize + size_new;
    writeBMP(temp, output);

    //Delete dynamic array
    delete[] image_new;
    for (int i = 0; i < h; i++)
    {
        delete[] image[i];
    }
    delete image;
    delete[] temp_data;
    delete temp;
}

```

Giải thích hàm:

- Tương tự như ở mục VII, hàm ở đây chỉ khác ở hàm countAverage24bit trả về giá trị là một biến Pixel_24bit.

IX. HÀM RESCALE ẢNH 32BPP

```

Pixel 32bit countAverage32bit(Pixel 32bit* a, int i cur, int j cur, int i increase, int j increase)
{
    unsigned char divide = unsigned char(i_increase) * unsigned char(j_increase);
    Pixel 32bit temp;
    int Sum_Blue = 0, Sum_Green = 0, Sum_Red = 0, Sum_A = 0;
    for (int i = i_cur; i < i_cur + i_increase; i++)
    {
        for (int j = j_cur; j < j_cur + j_increase; j++)
        {
            Sum_A += a[i][j].A;
            Sum_Blue += a[i][j].Blue;
            Sum_Green += a[i][j].Green;
            Sum_Red += a[i][j].Red;
        }
    }
    temp.Blue = Sum_Blue / divide;
    temp.Green = Sum_Green / divide;
    temp.Red = Sum_Red / divide;
    temp.A = Sum_A / divide;
    return temp;
}

void rescale_32bit(bmp* picture, const char* output, int s)
{
    bmp* temp = new bmp; // remember to delete
    *temp = *picture;

    // Get the data pixel size
    unsigned int w, h, size;
    w = picture->dbi.width;
    h = picture->dbi.height;

    //*****
    //Create a temporary pointer to go through every element of the pixeldata array
    char* temp_data;
    //*****

    //Create a Pixel Image Array ( 24bit ) ( this will be a 2D Array for easier manipulation )
    Pixel 32bit* image = new Pixel 32bit [ h ];
    for (int i = 0; i < h; i++)
    {
        image[i] = new Pixel 32bit[w];
    }
    //*****

    //Get the Data of 2D Pixel-Image
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            image[i][j].A = *(temp++);
            image[i][j].Blue = *(temp++);
            image[i][j].Green = *(temp++);
            image[i][j].Red = *(temp++);
        }
    }

    //Calculating the new size of the new image
    unsigned int h_new, w_new, padding_new;
    if (h % s == 0)
    {
        h_new = h / s;
    }
    else
    {
        h_new = h / s + 1;
    }
    if (w % s == 0)
    {
        w_new = w / s;
    }
    else
    {
        w_new = w / s + 1;
    }
    int size_new = w_new * h_new * 4; // 32bit

    //Creating the rescale image pixel data 2D Array
    Pixel 32bit* image_new = new Pixel 32bit[w_new * h_new];
    //*****

    int count = 0;
    for (int i = 0; i < h; i += s)
    {
        for (int j = 0; j < w; j += s)
        {
            if (i == (h - (h % s)) && h / s != 0)
            {
                image_new[count++] = countAverage32bit(image, i, j, h % s, s); // leftover byte
                continue;
            }
            if (j == (w - (w % s)) && w / s != 0)
            {
                image_new[count++] = countAverage32bit(image, i, j, s, w % s); // leftover byte
                continue;
            }
            image_new[count++] = countAverage32bit(image, i, j, s, s);
        }
    }

    //Put all the value into the temp-data
    temp_data = new char[size_new];
    temp = temp_data;
    count = 0;
    for (int i = 0; i < h_new; i++)
    {
        for (int j = 0; j < w_new; j++)
        {
            *(temp++) = image_new[count].A;
            *(temp++) = image_new[count].Blue;
            *(temp++) = image_new[count].Green;
            *(temp++) = image_new[count].Red;
            count++;
        }
    }
    temp->dbi.height = h_new;
    temp->dbi.width = w_new;
    temp->header.iLedsSize = 54 + (temp->header.DataOffset - 54) + size_new;
    writeBMP(temp, output);

    //delete dynamic array
    delete[] image_new;
    for (int i = 0; i < h; i++)
    {
        delete[] image[i];
    }
    delete image;
    delete[] temp_data;
    delete temp;
}

```

Giải thích hàm:

- Tương tự như ở mục VII, hàm ở đây chỉ khác ở hàm `countAverage32bit` trả về giá trị là một biến `Pixel_32bit`.

X. HÀM MAIN

```
#include "MyLib.h"

int main(int argc, char* argv[])
{
    bmp* a = nullptr;
    int check = readBMP(a, argv[2]);
    if (check == 0)
    {
        cout << "The image is not loaded successfully" << endl;
        return 0;
    }
    else
    {
        cout << "The image is loaded successfully" << endl;
    }
    short bpp = a->dib.bpp;
    if (strcmp(argv[1], "-conv") == 0 && argc == 4)
    {
        if (bpp == 24)
        {
            grayscale_24bit(a, argv[3]);
        }
        else if (bpp == 32)
        {
            grayscale_32bit(a, argv[3]);
        }
        else if (bpp == 8)
        {
            cout << "Ban khong the convert anh 8bpp ve 8bpp";
        }
    }
    else if (strcmp(argv[1], "-zoom") == 0 && argc == 5)
    {
        int scale = atoi(argv[4]);
        if (bpp == 8)
        {
            rescale_8bit(a, argv[3], scale);
        }
        else if (bpp == 24)
        {
            rescale_24bit(a, argv[3], scale);
        }
        else if (bpp == 32)
        {
            rescale_32bit(a, argv[3], scale);
        }
    }
    else
    {
        cout << "Command Line is not right, pls try again";
        system("pause");
        return 0;
    }
    delete[] a->colorpalette;
    delete[] a->data;
    delete a;
    system("pause");
    return 0;
}
```


Giải thích hàm main:

- Hàm tuân theo các quy tắc của tham số dòng lệnh và yêu cầu đề bài.
- Hàm có kiểm tra xem liệu ảnh có được load thành công hay không.