DIP HW1 Report

電子所 郭家均 313510171

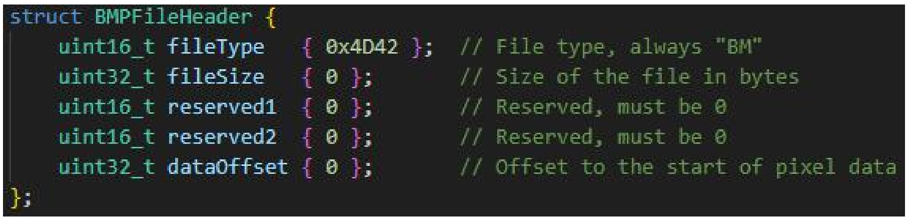
BMP format

BMP (Bitmap)為 windows 的圖像格式，其檔案分為以下四個部分:

1. File Header

Type	2 byte	Signature, typically “BM” (0x4D42)
Size	4 byte	The size of the file
Reserved 1	2 byte	Must be 0
Reserved 2	2 byte	Must be 0
Offset	4 byte	Offset between bitmap data and begin of BMP file

Implementation

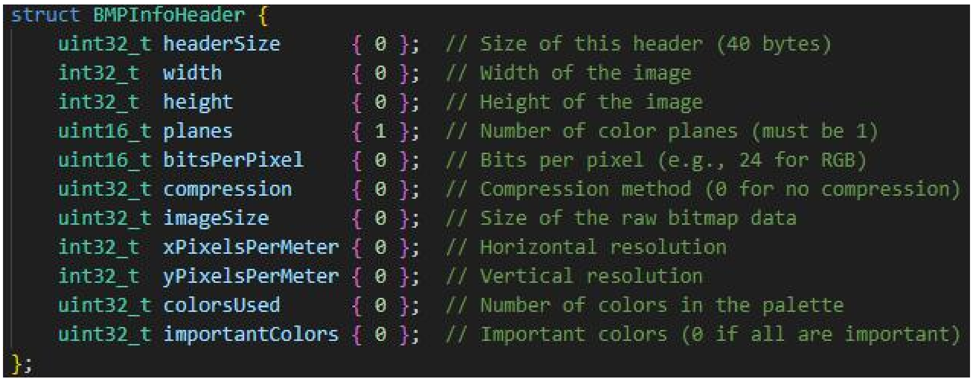


```
struct BMPFileHeader {
    uint16_t fileType   { 0x4D42 };  // File type, always "BM"
    uint32_t fileSize   { 0 };       // Size of the file in bytes
    uint16_t reserved1  { 0 };       // Reserved, must be 0
    uint16_t reserved2  { 0 };       // Reserved, must be 0
    uint32_t dataOffset { 0 };       // Offset to the start of pixel data
};
```

2. Info Header

Header size	4 byte	Size of this header (40 bytes)
Image Width	4 byte	Width of image pixel data
Image Height	4 byte	Height of image pixel data
Planes	2 byte	# of used color planes (must be 1)
Bits per Pixel	2 byte	Typically 1, 4, 8, 16, 24, 32 bits per pixel
Compression Method	4 byte	Set to 0 if no compression involve
Image size	4 byte	Size of the bitmap data
X Resolution	4 byte	Horizontal pixel per meter
Y Resolution	4 byte	Vertical pixel per meter
Colors in Palette	4 byte	Number of colors in the Palette (0 default)
Important Colors	4 byte	# of important colors

Implementation



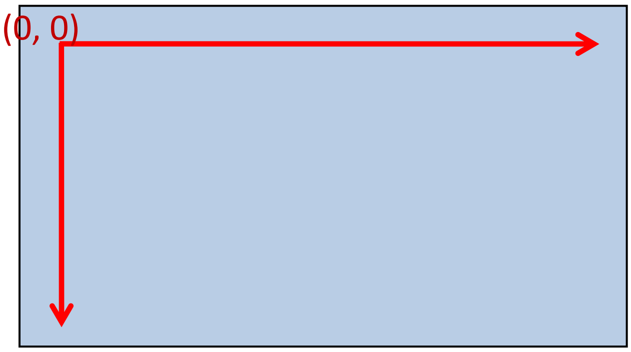
```
struct BMPInfoHeader {
    uint32_t headerSize      { 0 };  // Size of this header (40 bytes)
    int32_t  width           { 0 };  // Width of the image
    int32_t  height          { 0 };  // Height of the image
    uint16_t planes          { 1 };  // Number of color planes (must be 1)
    uint16_t bitsPerPixel    { 0 };  // Bits per pixel (e.g., 24 for RGB)
    uint32_t compression     { 0 };  // Compression method (0 for no compression)
    uint32_t imageSize       { 0 };  // Size of the raw bitmap data
    int32_t  xPixelsPerMeter { 0 };  // Horizontal resolution
    int32_t  yPixelsPerMeter { 0 };  // Vertical resolution
    uint32_t colorsUsed      { 0 };  // Number of colors in the palette
    uint32_t importantColors { 0 };  // Important colors (0 if all are important)
};
```

3. Raw Data

Contains the actual Bitmap pixel data: 1D vector, concatenate pixels in a {B G R} fashion

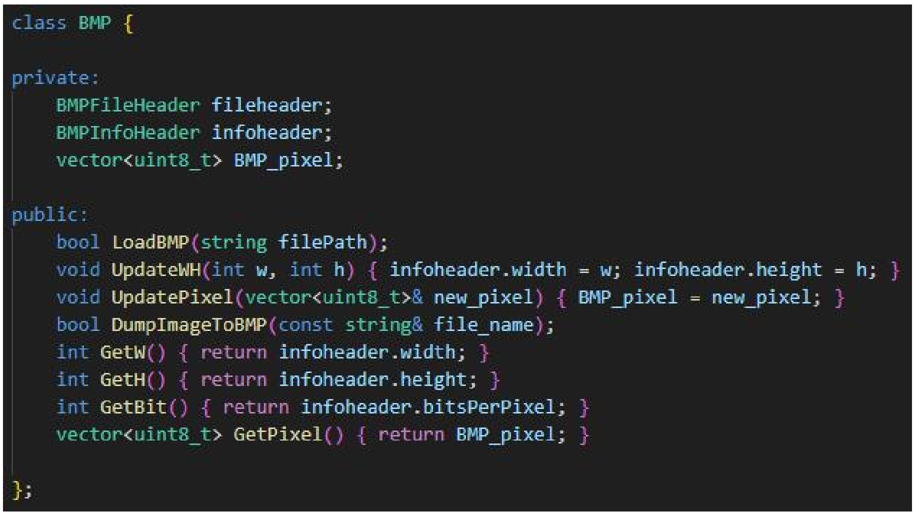
B	G	R	B	G	R	B	G	R	...	B	G	R

Same colors means those values represent the same pixel, scanning from left-to-right top-to-bottom fashion



4. Palette (optional) Not implemented, thus we skip this one

BMP class



```cpp
class BMP {

private:
    BMPFileHeader fileheader;
    BMPInfoHeader infoheader;
    vector<uint8_t> BMP_pixel;

public:
    bool LoadBMP(string filePath);
    void UpdateWH(int w, int h) { infoheader.width = w; infoheader.height = h; }
    void UpdatePixel(vector<uint8_t>& new_pixel) { BMP_pixel = new_pixel; }
    bool DumpImageToBMP(const string& file_name);
    int GetW() { return infoheader.width; }
    int GetH() { return infoheader.height; }
    int GetBit() { return infoheader.bitsPerPixel; }
    vector<uint8_t> GetPixel() { return BMP_pixel; }

};
```

LoadBMP	Load the BMP image into our class object
UpdateWH	Update the Width & Height of the image
UpdatePixel	Update the bitmap data
DumpImageToBMP	Dump the BMP class object to a image.bmp
GetW	Get the width of the image
GetH	Get the height of the image
GetBit	Get the bits per pixel
GetPixel	Get the bitmap pixel data

Load/Dump BMP image

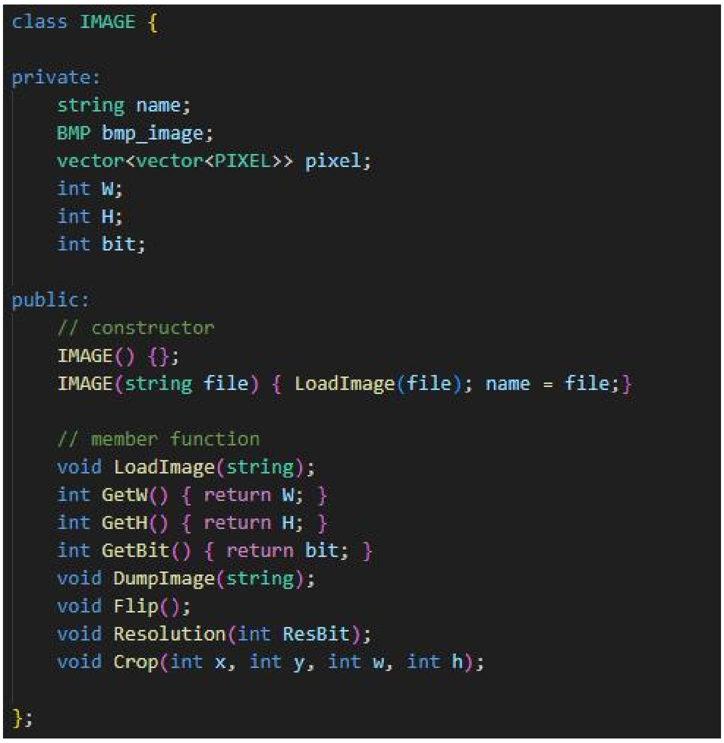
Load:

1. Use ifstream to load BMP image file
2. First get the file header using the size of the declared BMPFileHeader structure
3. Check the file type (signature), if it’s not 0x4D42, it is not a BMP image file
4. Get the info heade using the size of BMPInfoHeader
5. Since we only accept pixel to be 24 or 32 bits, any other size should be error
6. Use the seekg function and Offset from the info header to move to the pixel data location
7. Load in the pixel data by its bits per pixel and the width/height of the image
8. Image loaded successfully

Dumping the image is in similar fashion, and in reversed steps.

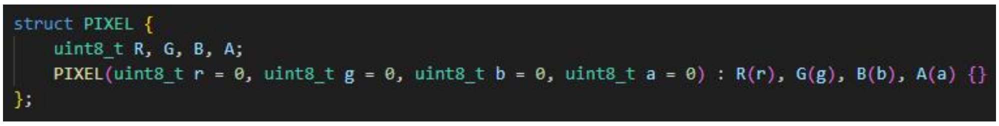
IMAGE class

To make our implementation more efficient, we use the class IMAGE to further simplify the process and transform the pixel data to be 2-dimensional.



```
class IMAGE {

private:
    string name;
    BMP bmp_image;
    vector<vector<PIXEL>> pixel;
    int W;
    int H;
    int bit;

public:
    // constructor
    IMAGE() {};
    IMAGE(string file) { LoadImage(file); name = file;}

    // member function
    void LoadImage(string);
    int GetW() { return W; }
    int GetH() { return H; }
    int GetBit() { return bit; }
    void DumpImage(string);
    void Flip();
    void Resolution(int ResBit);
    void Crop(int x, int y, int w, int h);

};
```

We let each operation become the member function of this class, and introduce Load/Dump function to transfer it to the BMP class object.

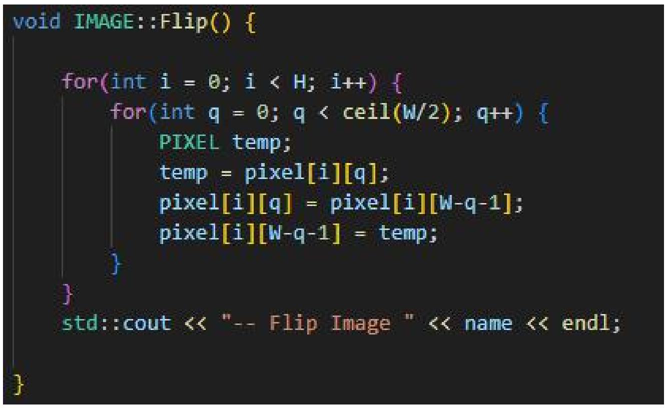


```
struct PIXEL {
    uint8_t R, G, B, A;
    PIXEL(uint8_t r = 0, uint8_t g = 0, uint8_t b = 0, uint8_t a = 0) : R(r), G(g), B(b), A(a) {}
};
```

The original 1D pixel vector is transfer to the 2D pixel vector as declared above. Variable A is the Alpha value of the 32bit-per-pixel BMP image, which we only preserved here.

Flip the image

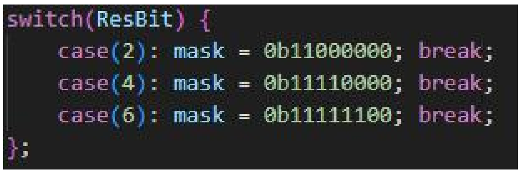
1. Use temporary PIXEL to save pixel
2. Exchange the pixel data value horizontally
3. Done



```
void IMAGE::Flip() {

    for(int i = 0; i < H; i++) {
        for(int q = 0; q < ceil(W/2); q++) {
            PIXEL temp;
            temp = pixel[i][q];
            pixel[i][q] = pixel[i][W-q-1];
            pixel[i][W-q-1] = temp;
        }
    }
    std::cout << "-- Flip Image " << name << endl;

}
```

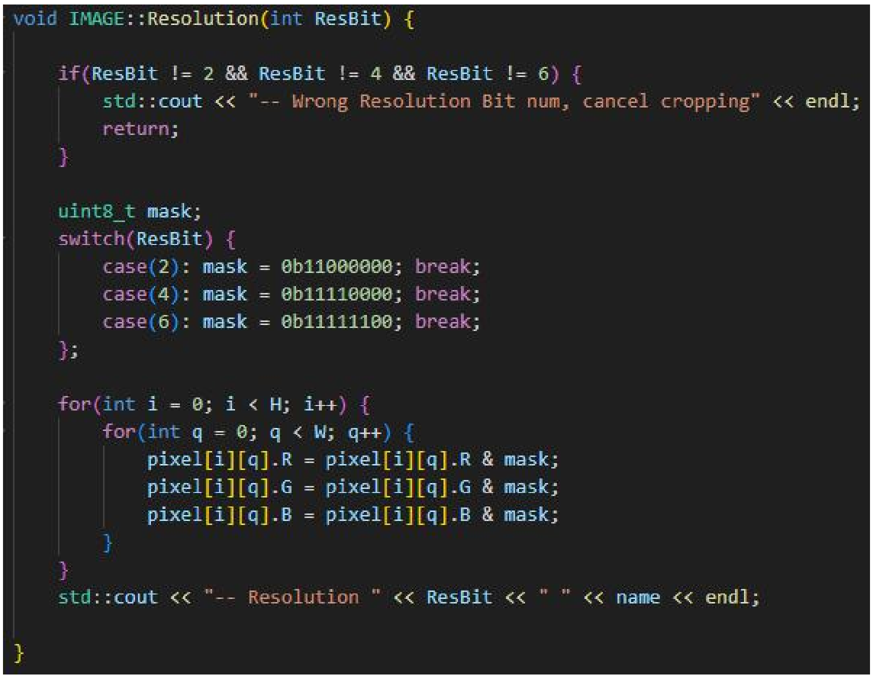
Quantization Resolution

1. Input the resolution bit size ResBit (2/4/6)
2. Create an 8-bit mask depends on ResBit



```
switch(ResBit) {
    case(2): mask = 0b11000000; break;
    case(4): mask = 0b11110000; break;
    case(6): mask = 0b11111100; break;
};
```

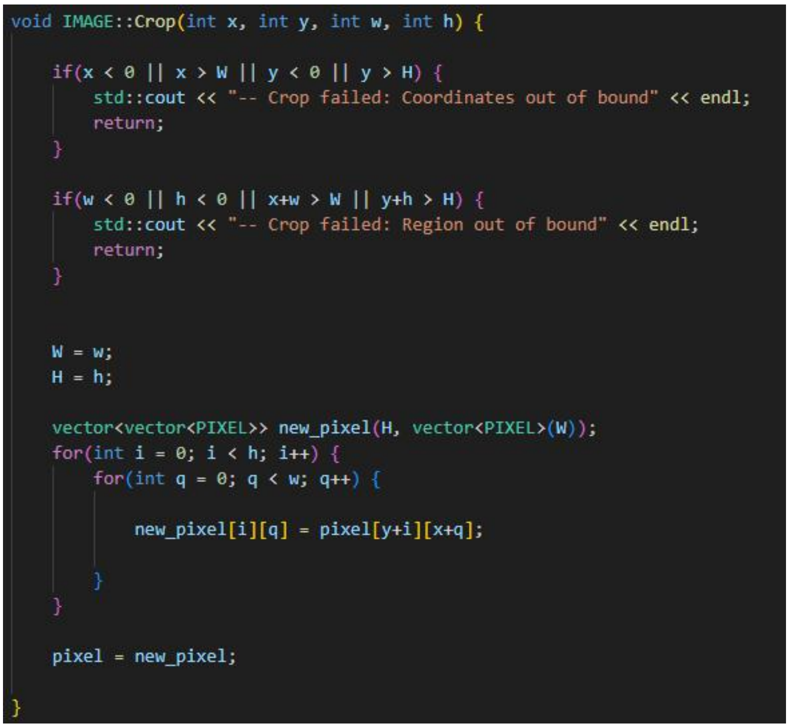
3. Iterate all pixel data, perform bit-wise AND (&) between each pixel and the mask
4. Done



```
void IMAGE::Resolution(int ResBit) {

    if(ResBit != 2 && ResBit != 4 && ResBit != 6) {
        std::cout << "-- Wrong Resolution Bit num, cancel cropping" << endl;
        return;
    }

    uint8_t mask;
    switch(ResBit) {
        case(2): mask = 0b11000000; break;
        case(4): mask = 0b11110000; break;
        case(6): mask = 0b11111100; break;
    };

    for(int i = 0; i < H; i++) {
        for(int q = 0; q < W; q++) {
            pixel[i][q].R = pixel[i][q].R & mask;
            pixel[i][q].G = pixel[i][q].G & mask;
            pixel[i][q].B = pixel[i][q].B & mask;
        }
    }
    std::cout << "-- Resolution " << ResBit << " " << name << endl;

}
```

Crop the image

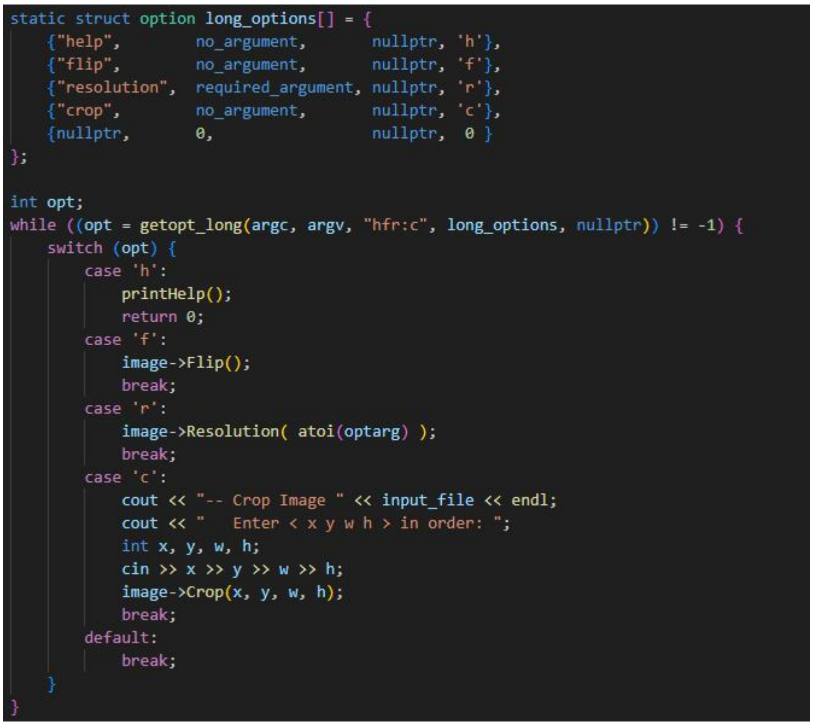
1. Check the boundary and input, if out of bound exit code.
2. Create new pixel vector based on new W, H
3. Extract the cropping region pixel to the new pixel vector
4. Set the image pixel vector to the new one



```
void IMAGE::Crop(int x, int y, int w, int h) {

    if(x < 0 || x > W || y < 0 || y > H) {
        std::cout << "-- Crop failed: Coordinates out of bound" << endl;
        return;
    }

    if(w < 0 || h < 0 || x+w > W || y+h > H) {
        std::cout << "-- Crop failed: Region out of bound" << endl;
        return;
    }


    W = w;
    H = h;

    vector<vector<PIXEL>> new_pixel(H, vector<PIXEL>(W));
    for(int i = 0; i < h; i++) {
        for(int q = 0; q < w; q++) {

            new_pixel[i][q] = pixel[y+i][x+q];

        }
    }

    pixel = new_pixel;

}
```

Options

5. To make our code more flexible, introduce option for the user to define each process steps



```
static struct option long_options[] = {
    {"help",        no_argument,       nullptr, 'h'},
    {"flip",        no_argument,       nullptr, 'f'},
    {"resolution",  required_argument, nullptr, 'r'},
    {"crop",        no_argument,       nullptr, 'c'},
    {nullptr,       0,                 nullptr,  0 }
};

int opt;
while ((opt = getopt_long(argc, argv, "hfr:c", long_options, nullptr)) != -1) {
    switch (opt) {
        case 'h':
            printHelp();
            return 0;
        case 'f':
            image->Flip();
            break;
        case 'r':
            image->Resolution( atoi(optarg) );
            break;
        case 'c':
            cout << "-- Crop Image " << input_file << endl;
            cout << "    Enter < x y w h > in order: ";
            int x, y, w, h;
            cin >> x >> y >> w >> h;
            image->Crop(x, y, w, h);
            break;
        default:
            break;
    }
}
```