

# 1. 前言

BMP 是英文 Bitmap（位图）的简写（本文简称位 BMP 位图），它是 Windows 操作系统中的标准图像文件格式，能够被多种 Windows 应用程序所支持。BMP 图形文件是 Windows 采用的图形文件格式，在 Windows 环境下运行的所有图象处理软件都支持 BMP 图象文件格式。Windows 系统内部各图像绘制操作都是以 BMP 为基础的。

**关键词：**BMP 位图文件结构分析、BMP 位图的读取、BMP 位图的保存、BMP 位图的显示

## 2. 实验内容与相关平台

### 2.1 实验内容

- 对 BMP 位图的文件结构进行分析
- 编写程序，实现对一张 24 位彩色位图进行读取、显示、保存

### 2.2 实验的相关平台与工具

Notepad++（用于分析图像的文件结构）、Vs code、C++

## 3. BMP 位图文件结构分析

在对 BMP 位图文件进行结构分析前，我们几个关键点需要提前知道：

- ① 在 BMP 文件中，数据存储采用**小端方式(little endian)**，即“低地址存放低位数据，高地址存放高位数据”。
- ② 以下所有分析均以**字节为单位**进行。

### 3.1 位图的文件头分析

位图文件头主要用于识别位图文件，共占 14 个字节。图 3-1 是位图文件头结构的定义：

```
01. 1. typedef struct tagBITMAPFILEHEADER {
02. 2.     WORD    bfType;
03. 3.     DWORD    bfSize;
04. 4.     WORD    bfReserved1;
05. 5.     WORD    bfReserved2;
06. 6.     DWORD    bfOffBits;
07. 7. } BITMAPFILEHEADER;
```

图 3-1

结构体中的字段含义如表 3-2 所示：

字段	字节数	含义
bfType	2	声明文件的类型，该值必须为 0x4D42，即字符'BM'。表示这是 Windows 支持的位图格式。 【注】该值也可以设置位'BA','CI','CP'等不同格式，但由于因为 OS/2 系统并没有被普及开，所以在编程时，只需判断第一个标识“BM”即可。
bfSize	4	声明 Bmp 文件的大小，单位是字节
bfReserved1	2	保留字段，必须设置为 0
bfReserved2	2	保留字段，必须设置为 0
bfOffBits	4	声明从文件头开始到实际的图象数据之间的字节的偏移量，可以用这个偏移值迅速的从文件中读取到位数据。

表 3-2

用 Notepad++打开 BMP 图像文件“lena-单色位.bmp”，如下图 3-3 所示。可见红框 1 中，第 1-2 字节数据为 0x4d42，为 BMP 位图的固定标识。在红框 2 中，第 3-6 字节数据为 0x00008d8e，即 36238 字节，与查看文件属性中的图片大小的数值一致。

在红框 3 中，这里的数据为 0x0000003e，即 62 字节，表示数据区位于从文件开始往后数的 62 字节处。

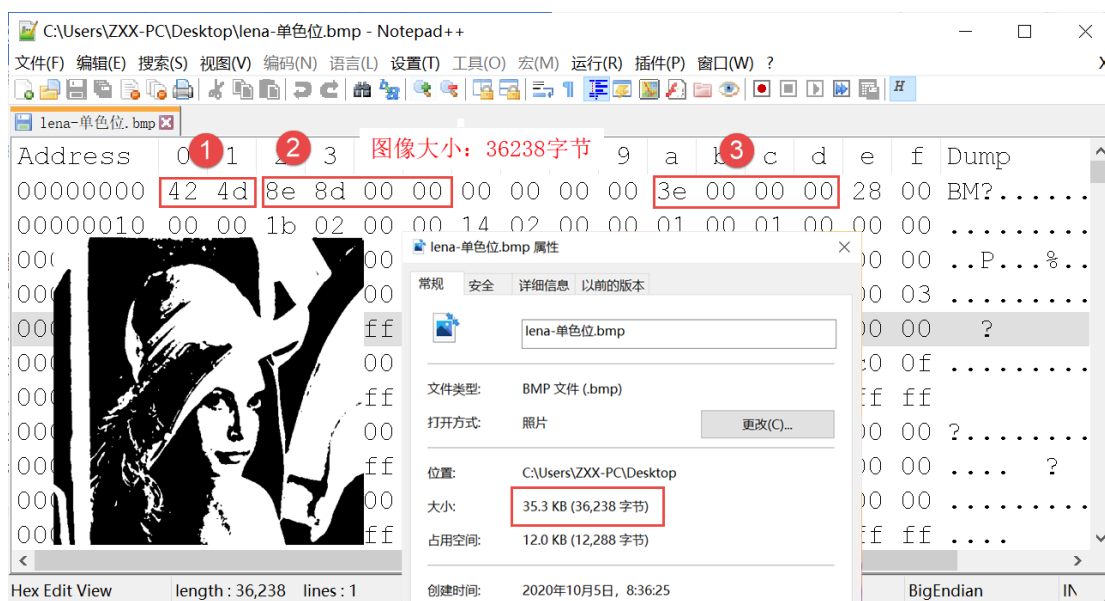


图 3-3

## 3.2 位图的信息头分析

BITMAPINFO 段由两部分组成：BITMAPINFOHEADER 结构体和 RGBQUAD 结构体，其中的 BITMAPINFOHEADER 结构体表示位图信息头。同样地，Windows 为位图信息头定义了如下结构体，如下图 3-4 所示：

```
01. typedef struct tagBITMAPINFOHEADER {
02.     DWORD biSize;
03.     LONG biWidth;
04.     LONG biHeight;
05.     WORD biPlanes;
06.     WORD biBitCount;
07.     DWORD biCompression;
08.     DWORD biSizeImage;
09.     LONG biXPelsPerMeter;
10.     LONG biYPelsPerMeter;
11.     DWORD biClrUsed;
12.     DWORD biClrImportant;
13. } BITMAPINFOHEADER;
```

图 3-4

结构体中的字段含义如下表 3-5 所示：

字段	占字节数	含义
biSize	4	声明 BITMAPINFOHEADER 所需要的字节数
biWidth	4	声明图片的宽度，单位是像素
biHeight	4	声明图片的高度，单位是像素
biPlanes	2	声明目标设备说明位面数，其值将总是被设为 1
biBitCount	2	声明单位像素的位数，表示 Bmp 图像的颜色位数，如 24 位图，32 位图
biCompression	4	声明图像的压缩属性，由于 bmp 图片是不压缩的，该值等于 0
biSizeImage	4	声明 Bmp 图像数据区的大小
biXPelsPerMeter	4	声明图像的水平分辨率
biYPelsPerMeter	4	声明图像的垂直分辨率
biClrUsed	4	声明使用了颜色索引表的数量
biClrImportant	4	声明重要的颜色的数量，等于 0 时表示所有颜色都很重要

表 3-5

继续用 Notepad++ 查看 BMP 图片，如下图 3-6 所示。可见红框 1 所示的数据表示 biSize 字段的值 0x0000 0028=40，表示位图信息头的大小为 40 字节。红框 2 与 3 表示图像的宽度与高度，其中 0x0000 021b = 539 像素，0x0000 0214 = 532 像素。红框 4 处表示图像的位深度，因为这是一个黑白图像，所以位深度为 1。

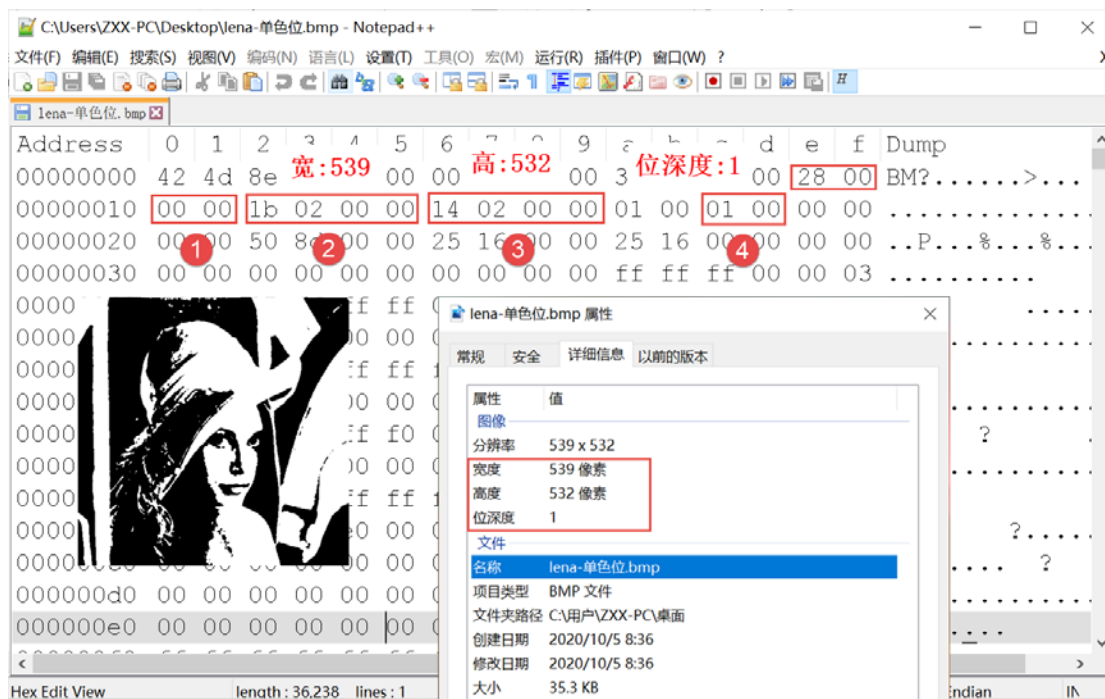


图 3-6

若打开的是 24 位深度的图片，可见该出数据的值为 0x0018，代表颜色深度为 24，如下图所示。

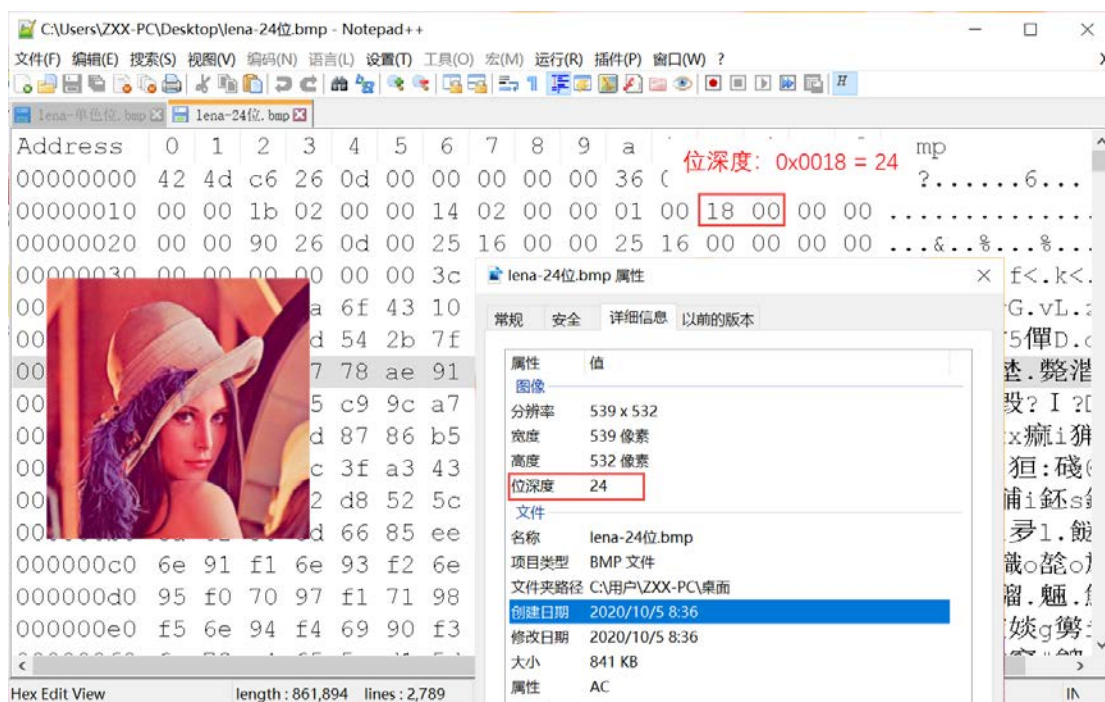


图 3-7

继续分析文件，如下图所示 3-8 所示。红框 5 处声明了 BMP 图像的数据区大小，0x00008d50 = 36176 字节。红框 6 处定义了图像的水平分辨率和垂直分辨率，红框 7 处定义了使用彩色表的索引值的数量，当该值为 0 时，表示使用所有调色板项。

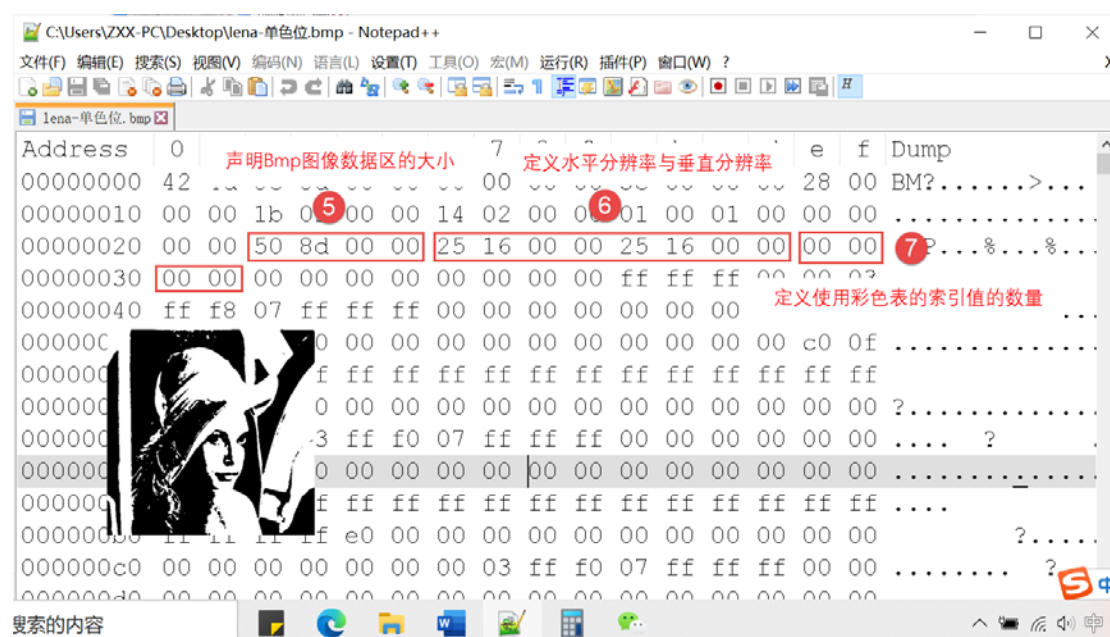


图 3-8

### 3.3 调色板分析

调色板的数据由 RGBQUAD 结构体项组成，该结构体由 4 个字节型数据组成，所以一个 RGBQUAD 结构体只占用 4 字节空间，从左到右每个字节依次表示(蓝色，绿色，红色，未使用)。调色板一般是针对 16 位以下的图像设置的，对于 16 位及以上的 BMP 格式图像，其位图像素数据是直接对应像素的 RGB 颜色值进行描述，因此省去了调色板。对于 16 位以下的 BMP 格式图像，其位图像素数据中记录的是调色板的索引值。

调色板的结构体定义如下图所示：

```

01.  typedef struct tagRGBQUAD {
02.     BYTE    rgbBlue;
03.     BYTE    rgbGreen;
04.     BYTE    rgbRed;
05.     BYTE    rgbReserved;
06.  } RGBQUAD;

```

图 3-9

分析图像的第 55-62 个字节，该出是图像的彩色表项，由于现在使用的图像是单色图，只有黑白两种颜色，所以调色板中也只有两项，对应着黑色和白色。如下图 3-10 所示。

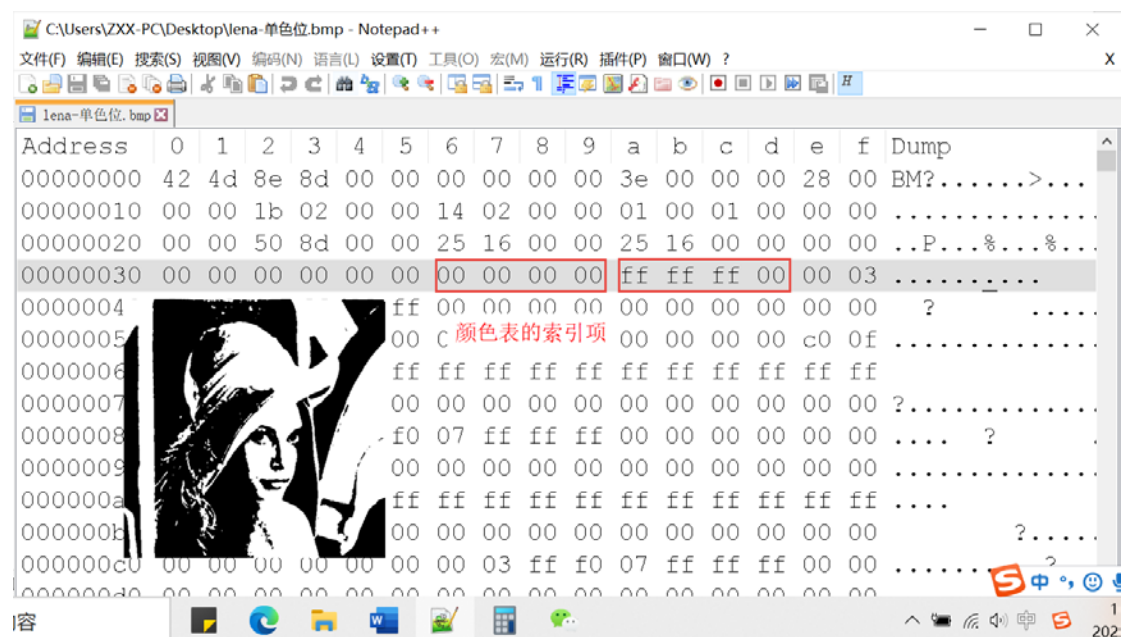


图 3-10

### 3.4 位图数据分析

位图数据记录了位图的每一个像素值，记录顺序是在扫描行内是从左到右，扫描行之间是从下到上。根据不同的位图，位图数据所占据的字节数也是不同的。比如，对于 24 位位图，每三个字节表示一个像素。对于本案例中的单色图，一个字节则可以对应八个像素点的像素值。

根据图像提供的位图数据，可以得知每个像素点的值，以此绘制图像。

如下图 3-11 所示，位图数据共有 36176 字节，位图文件头与位图信息头共 54 字节，再加上彩色表的两个索引项共 8 个字节，可以得知该图像共 36238 字节。此数据与用 Window 资源管理器直接查看图像的大小一致。



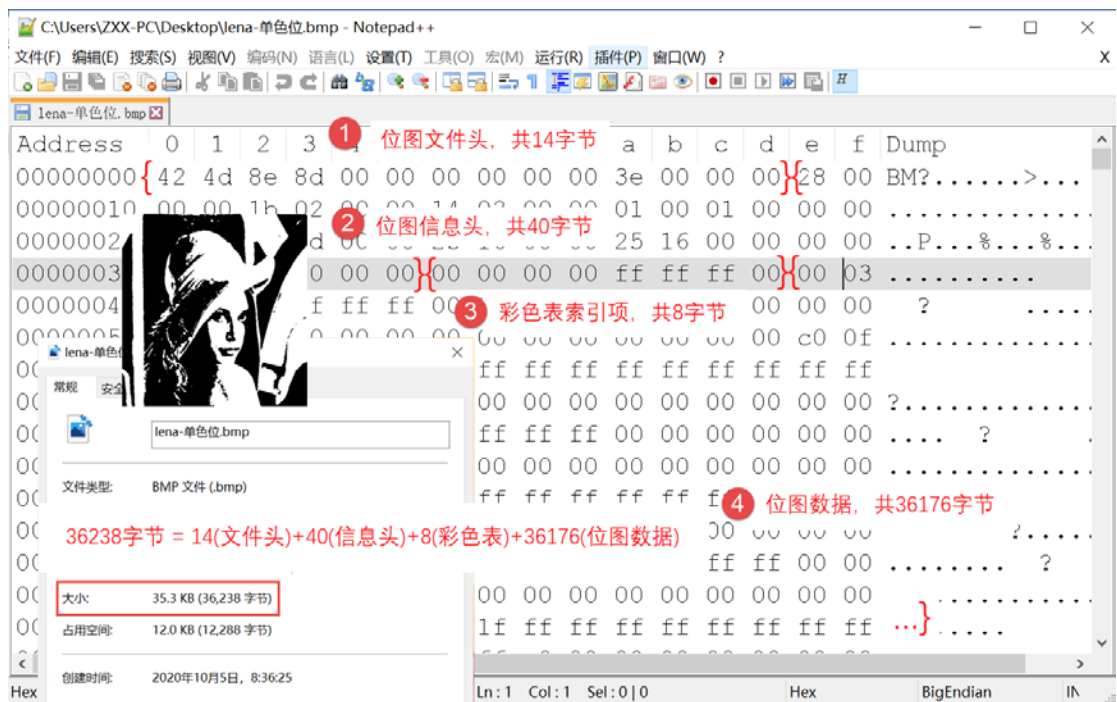


图 3-11

在位图数据中，有一个关键点需要特别注意：**BMP 存储格式要求每行的字节数必须是 4 的倍数**。若某行的字节数不是 4 的倍数，需要额外添加字符'0'凑够到 4 的倍数。在对位图数据进行读写时，这一点需要特别留意，否则无法进行位图图像的读写。

#### 4. 使用 C++实现对位图文件的读写、显示

在这一步中，将会用 C++实现对位图文件的读写，显示操作。其中位图文件的位深度为 24 位。

另外，在程序中自定义了一个新的结构体 `ImgInfo`，里面包含了位图文件头 `BITMAPFILEHEADER`、位图信息头 `BITMAPINFOHEADER`，还有一个二维数组 `imgData`，用于存放像素值信息。如下图 4-1 所示，这样做的目的是使用二维数组可以更方便地对图像的像素点进行操作。





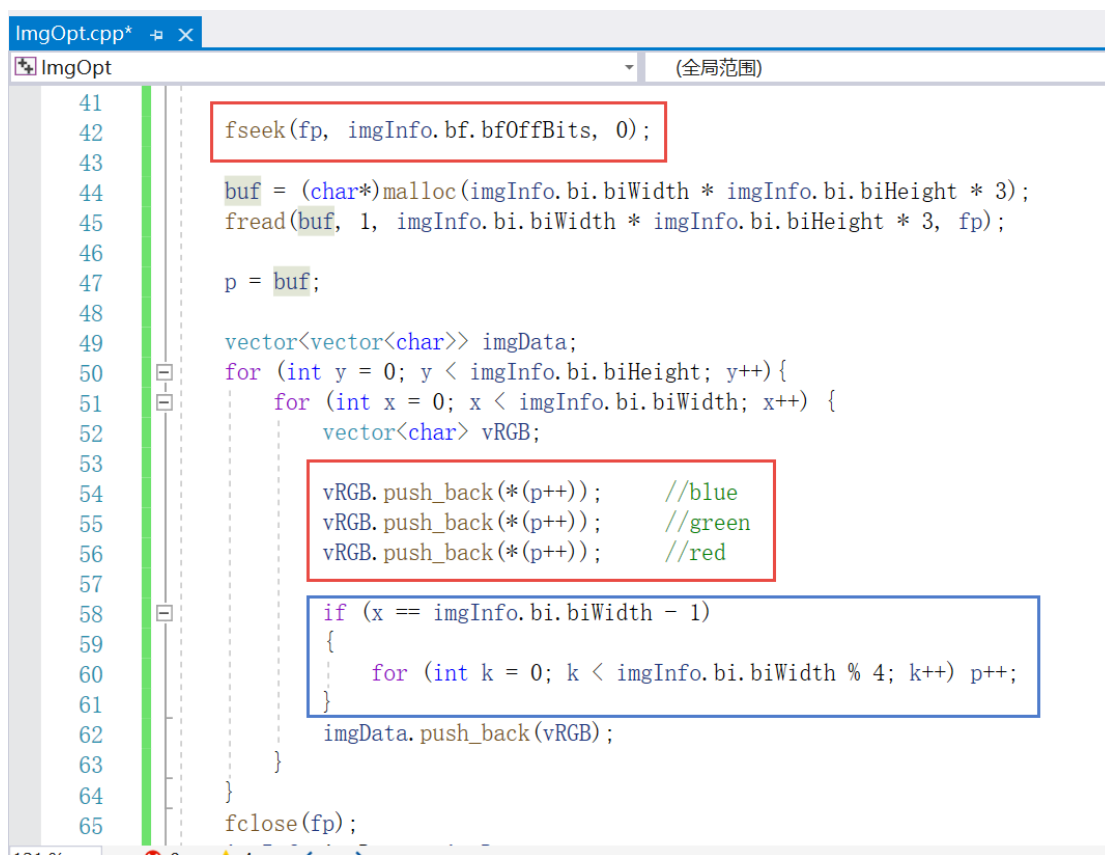


图 4-3

## 4.2 在控制台上显示位图图像

在控制台上显示位图图像，主要由 showBitmap 函数实现。根据结构体 ImgInfo 中的 imgData 字段，我们可以很轻易地获取图像的像素值信息，并通过 SetPixel 函数将像素值显示在特定的位置上，关键代码如下图 4-4 所示。需要注意的是，**BMP 位图的像素数据存储方式是行内从左到右，行间从下到上**，因此在编程时需要考虑清楚像素点与其对应的坐标位置。

```

void showBitmap(ImgInfo imgInfo) {
    HWND hWindow; //窗口句柄
    HDC hDc; //绘图设备环境句柄
    int yOffset = 150;
    hWindow = GetForegroundWindow();
    hDc = GetDC(hWindow);

    int posX, posY;
    for (int i = 0; i < imgInfo.imgData.size(); i++) {
        char blue = imgInfo.imgData.at(i).at(0);
        char green = imgInfo.imgData.at(i).at(1);
        char red = imgInfo.imgData.at(i).at(2);

        posX = i % imgInfo.bi.biWidth;
        posY = imgInfo.bi.biHeight - i / imgInfo.bi.biWidth + yOffset;
        SetPixel(hDc, posX, posY, RGB(red, green, blue));
    }
}

```

图 4-4

## 4.3 位图文件的保存

位图文件的保存，主要在 saveBitmap 函数中实现，如下图 4-5 所示。与位图文件的读取类似，先采用 fwrite 函数实现对位图文件头和位图信息头的写入，再遍历像素点将像素值写入。

同样地，在每写完一行位图数据后，需要填充'0'字符，实现 4 字节对齐的存储方式。

```
88
89 void saveBitmap(ImgInfo imgInfo) {
90     FILE* fpw;
91     fopen_s(&fpw, saveImgPath.c_str(), "wb");
92     fwrite(&imgInfo.bf, sizeof(BITMAPFILEHEADER), 1, fpw); //写入文件头
93     fwrite(&imgInfo.bi, sizeof(BITMAPINFOHEADER), 1, fpw); //写入文件头信息
94
95     int size = imgInfo.bi.biWidth * imgInfo.bi.biHeight;
96     for (int i = 0; i < size; i++) { //写入像素数据
97         fwrite(&imgInfo.imgData.at(i).at(0), 1, 1, fpw);
98         fwrite(&imgInfo.imgData.at(i).at(1), 1, 1, fpw);
99         fwrite(&imgInfo.imgData.at(i).at(2), 1, 1, fpw);
100
101         //填充'0'字符，实现4字节对齐的存储方式
102         if (i % imgInfo.bi.biWidth == imgInfo.bi.biWidth - 1) {
103             char ch = '0';
104             for (int j = 0; j < imgInfo.bi.biWidth % 4; j++) {
105                 fwrite(&ch, 1, 1, fpw);
106             }
107         }
108     }
109     fclose(fpw);
110     cout << "已保存图像至: " + saveImgPath << endl;
111 }
112
```

图 4-5

## 4.4 实验效果

如下图 4-6 所示，在运行程序后，会将图像数据读出，然后在控制台上显示图像，并将图像保存到桌面上。

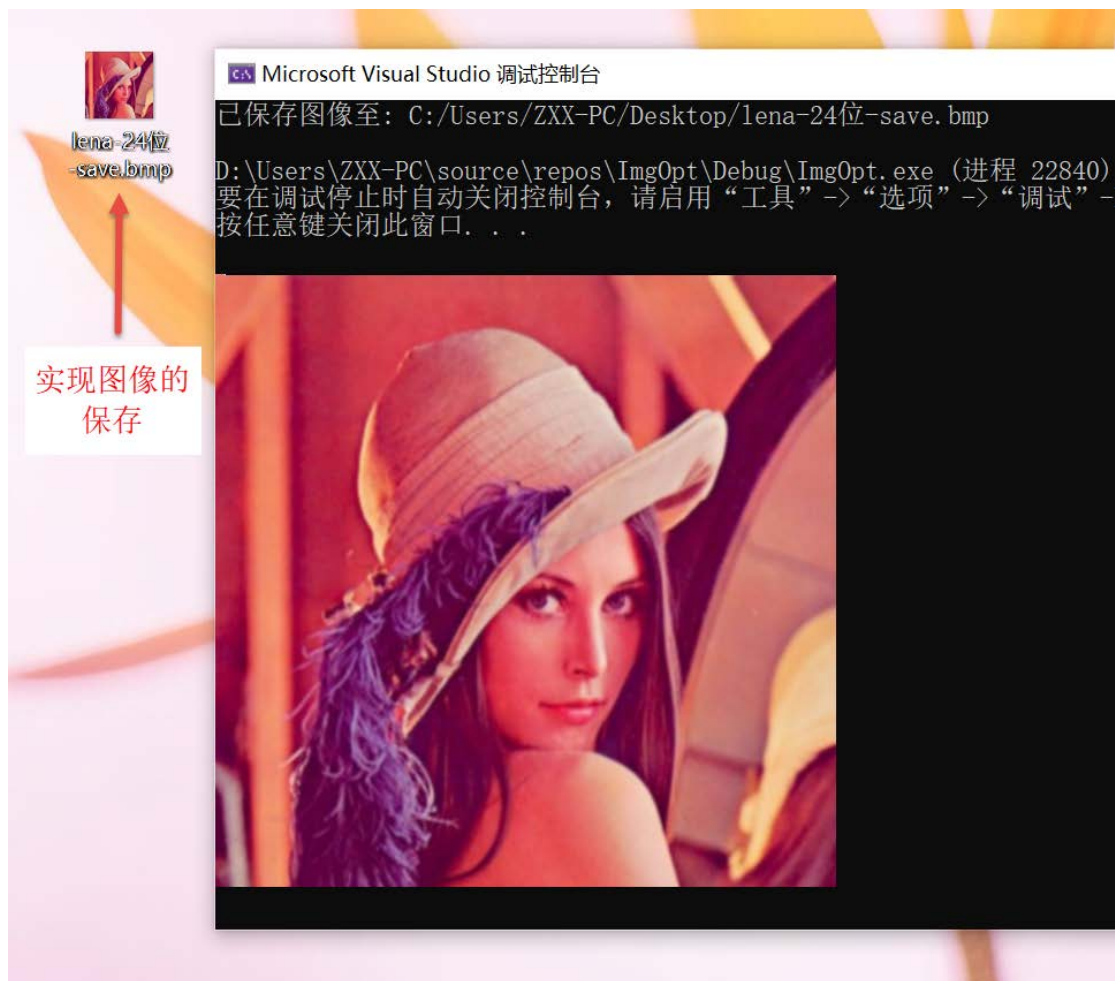


图 4-6