



成	
绩	

廣東工業大學

实验报告

课 程 名 称 现代图像处理技术

题 目 名 称 位图(BMP)文件结构分析及
文件读取 、显示与保存

学 院 计算机学院

专 业 班 级 电子信息专硕 1 班

学 生 姓 名 梁增国

学 号 2112005119

指 导 老 师 战荫伟

日 期 2020 年 10 月 7 日

目 录

1. 摘要.....	1
2. 实验内容与相关平台.....	1
2.1 实验内容	1
2.2 实验的相关平台与工具.....	1
3. BMP 位图文件结构分析	1
3.1 位图的文件结构	1
3.2 位图的文件头分析.....	2
3.3 位图的信息头分析.....	3
3.4 调色板分析.....	5
3.5 位图数据分析	7
4. 使用 C++实现对位图文件的读写、显示	8
4.1 位图文件的读取	8
4.2 在控制台上显示位图图像.....	9
4.3 位图文件的保存	10
4.4 实验效果	11
5. 总结.....	11
6. 参考文章.....	12

1. 摘要

BMP 是英文 Bitmap（位图）的简写，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 图像文件是 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 图像文件格式。Windows 系统内部各图像绘制操作都是以 BMP 为基础的。

本实验将会对 BMP 位图的文件结构进行分析，并实现 BMP 位图的读取、显示和保存。

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. 实验内容与相关工具

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 24 位彩色位图进行读取、显示、保存操作

2.2 实验的相关工具

Notepad++（用于分析 BMP 图像的文件结构）、Vs code、C++

3. BMP 位图文件结构分析

在本节中，我们将要对 BMP 位图的文件结构进行分析。在此之前，我们需要事先了解两个关键点：

- 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”。
- 以下所有分析均以**字节为存储单位**。

3.1 位图的文件结构

位图的文件结构如表 3-1 所示，位图的数据包括四项，分别是：**位图文件头**、**位图信息头**、**调色板**和**位图数据**。

表 3-1 位图的文件格式

位图文件头 BITMAPFILEHEADER
位图信息头 BITMAPINFOHEADER

调色板 Palette
位图数据 ImageData

3.2 位图的文件头分析

位图文件头主要用于识别位图文件，以及记录文件的大小、位图数据位置等信息，共占 14 个字节。图 3-2 是位图文件头结构的定义，位图文件头的字段含义如表 3-3 所示。

```
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 3-2

表 3-3 位图头文件的字段以及含义

字段	字节数	含义
bfType	2	声明位图文件的类型，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。因此在编程时，只需判断第一个标识是否为“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图像像素数据之间的字节偏移量，实际中可以根据该偏移值迅速地文件中读取到图像的像素数据

用 Notepad++打开 BMP 图像文件“lena-单色位.bmp”，如下图 3-4 所示。可见红框 1 中，第 1-2 字节数据为 0x4d42，是 BMP 位图的固定标识。在红框 2 中，第 3-6 字节数据为 0x00008d8e，表示图像的大小为 36238 字节，可见该值与在 Window 资源管理器中查看文件属性中的图像大小的是是一致的。

在红框 3 中，此处的数据为 0x0000003e，即 62 字节，表示位图数据位于从文件开始往后数的第 62 字节处。



图 3-4

3.3 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 3-5 所示。位图信息头的字段含义如下表 3-6 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 3-5

表 3-6 位图信息头的字段以及含义

字段	占字节数	含义
biSize	4	声明 BITMAPINFOHEADER 占用的字节数
biWidth	4	声明图像的宽度，单位是像素
biHeight	4	声明图像的高度，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	声明图像压缩属性, 由于 BMP 图像是不压缩, 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

继续用 Notepad++ 分析 BMP 图像的文件结构, 如下图 3-7 所示。可见红框 1 所示为 biSize 字段, 它的值为 0x00000028=40, 表示位图信息头的大小为 40 字节。红框 2 与 3 所示的数据表示图像的宽度与高度, 对应的值为 0x0000021b = 539 像素, 0x00000214 = 532 像素。

红框 4 处表示图像的位深度, 因为这是一张黑白图像, 所以位深度为 1。

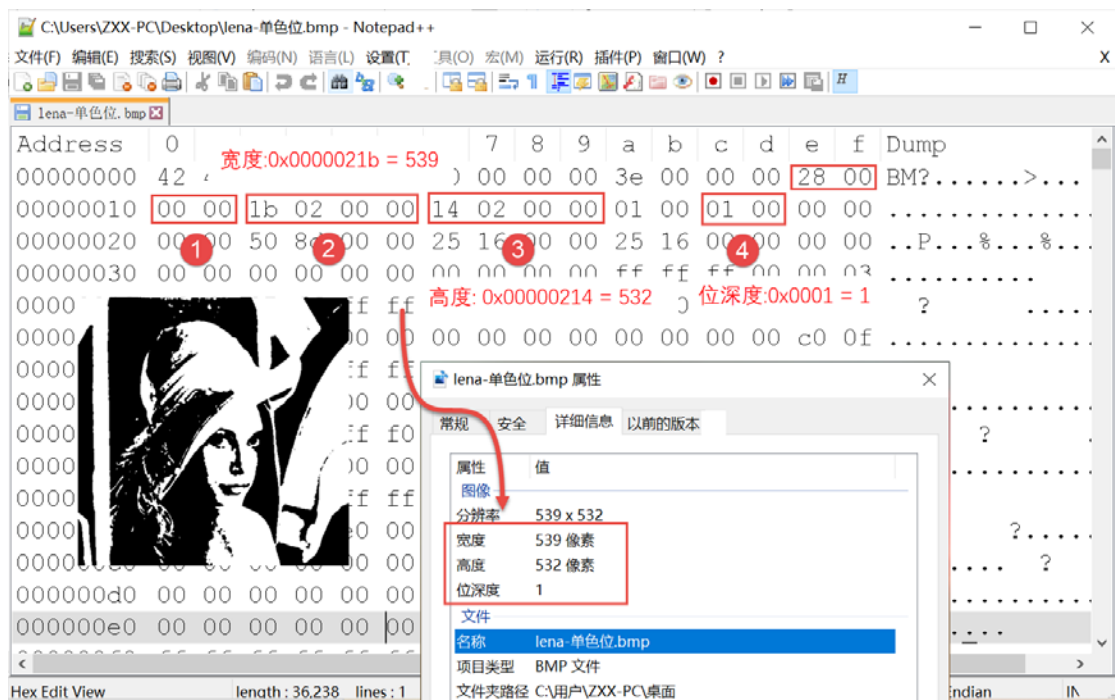


图 3-7

若打开的是 24 位深度的图像, 可见该字段的值为 0x0018, 代表颜色深度为 24, 如下图 3-8 所示。

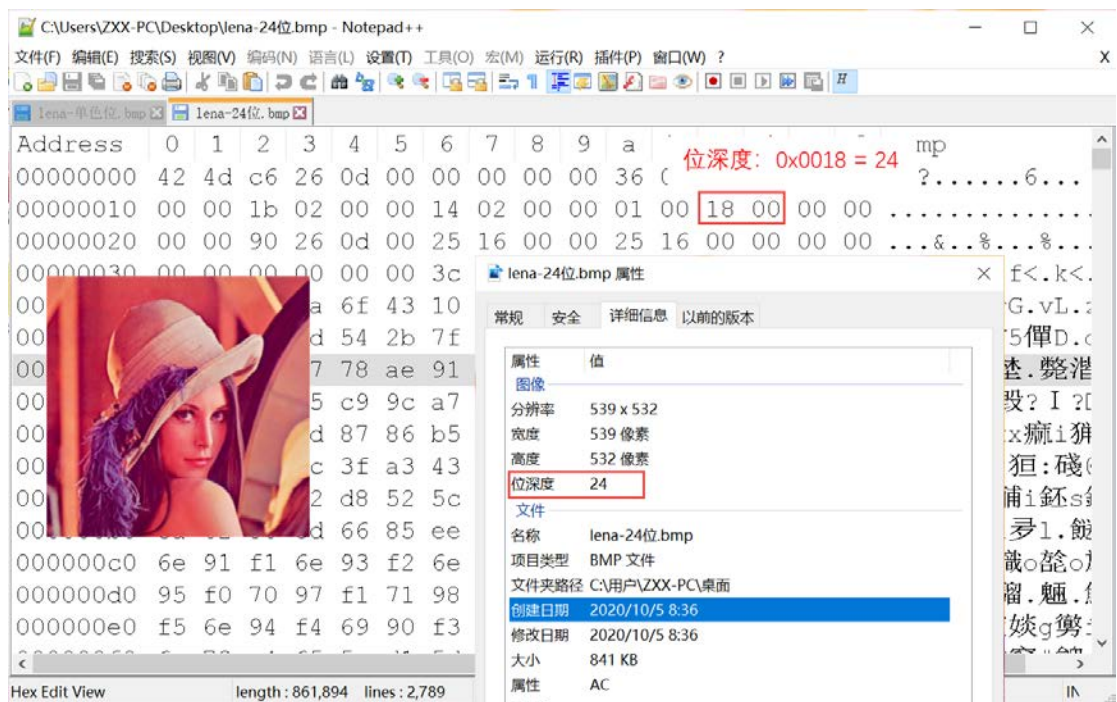


图 3-8

继续分析文件，如下图 3-9 所示。红框 5 处声明了 BMP 图像的数据区大小，即 $0x00008d50 = 36176$ 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。

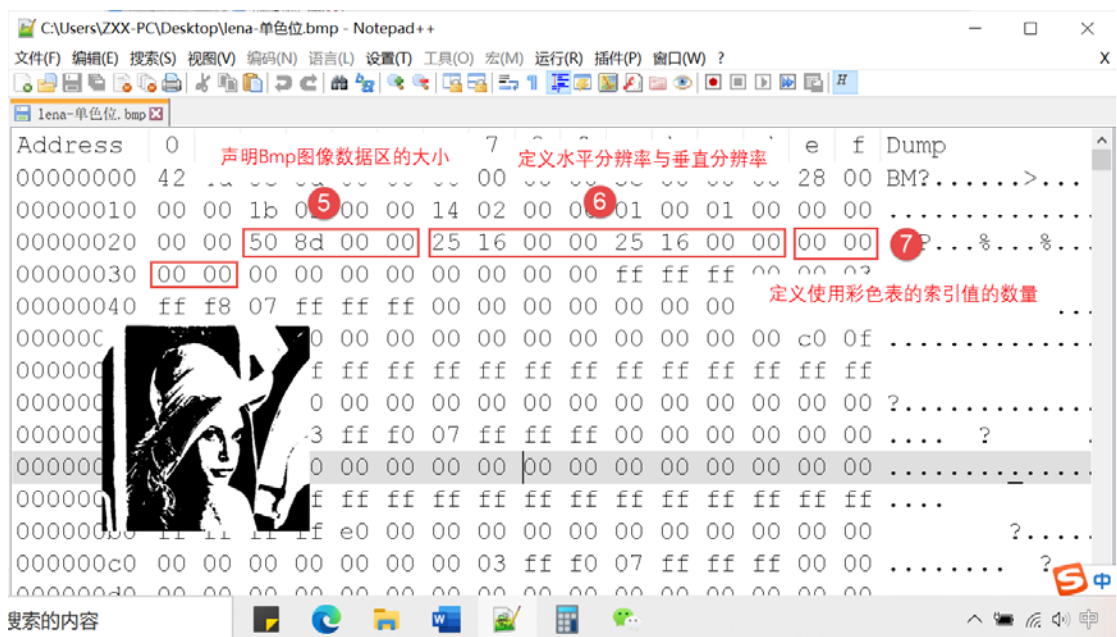


图 3-9

3.4 调色板分析

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。调色板的作用是，当图像的位深度值比较小时，通过调色板记录所有的颜色值，而位图数据则存储调色板的索引项，因此达到节省存储空间的效果。

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)。调色板的结构体定义如下图 3-10 所示：

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

图 3-10

分析图像的第 55-62 个字节，该处声明的是图像的彩色表项，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 3-11 所示。

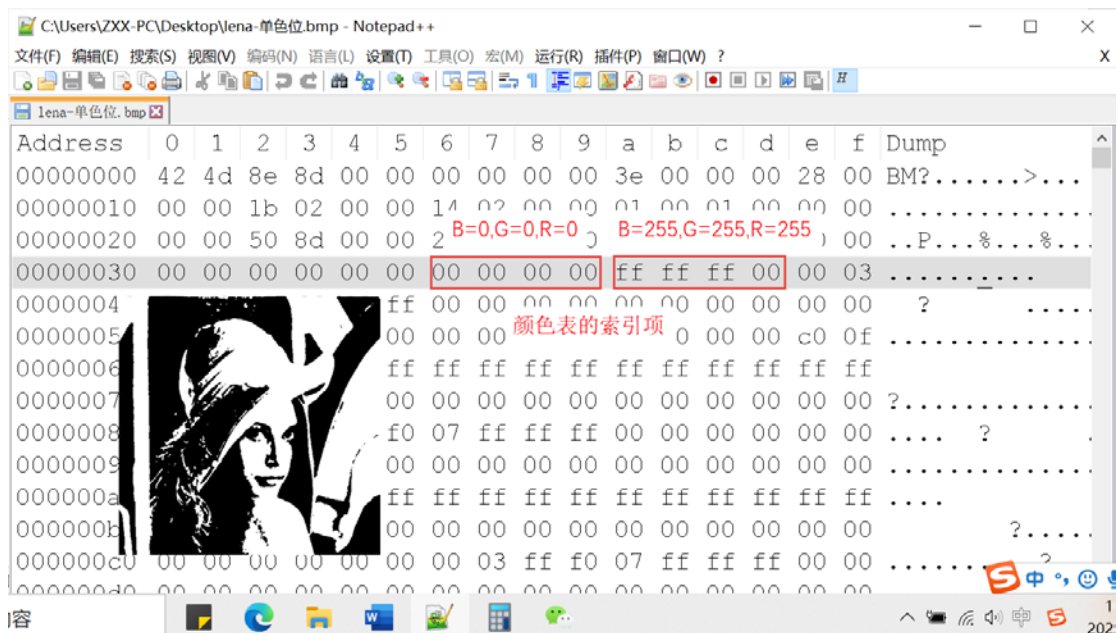


图 3-11

接下来，我们分析位深度大于 16 位的 BMP 图像的调色板。

我们用 Notepad++ 打开一张位深度为 24 的 BMP 图像，如下图 3-12 所示。红框 1 处为位图文件头的 `bfOffBits` 字段，值为 54 字节，表示从文件头起始到位图数据之间的字节的偏移量 54 字节。

红框 2 处的字段为位图信息头的 biSize 字段，值为 40 字节。观察两组数据，位图的文件头固定为 14 字节，加上信息头的 40 字节因此总字节数为 54 字节，正等于 bfOffBits 字段的偏移量。

由此得知，位深度为 24 的 BMP 图像是没有调色板数据的。

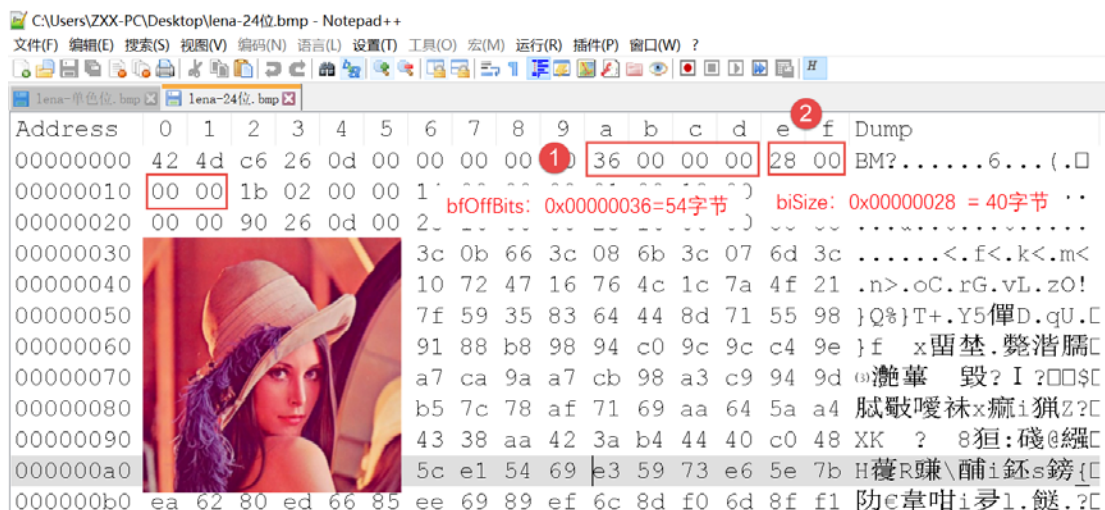


图 3-12

3.5 位图数据分析

位图数据记录了位图中每一个像素的像素值，存储的规则是：行内从左到右，行间从下到上。位深度不同的位图，位图数据所占据的字节数也是不同的。比如，对于 24 位的位图，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像素值，以此绘制图像。

如下图 3-12 所示，位图数据共有 36176 字节，位图文件头与位图信息数据共 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。

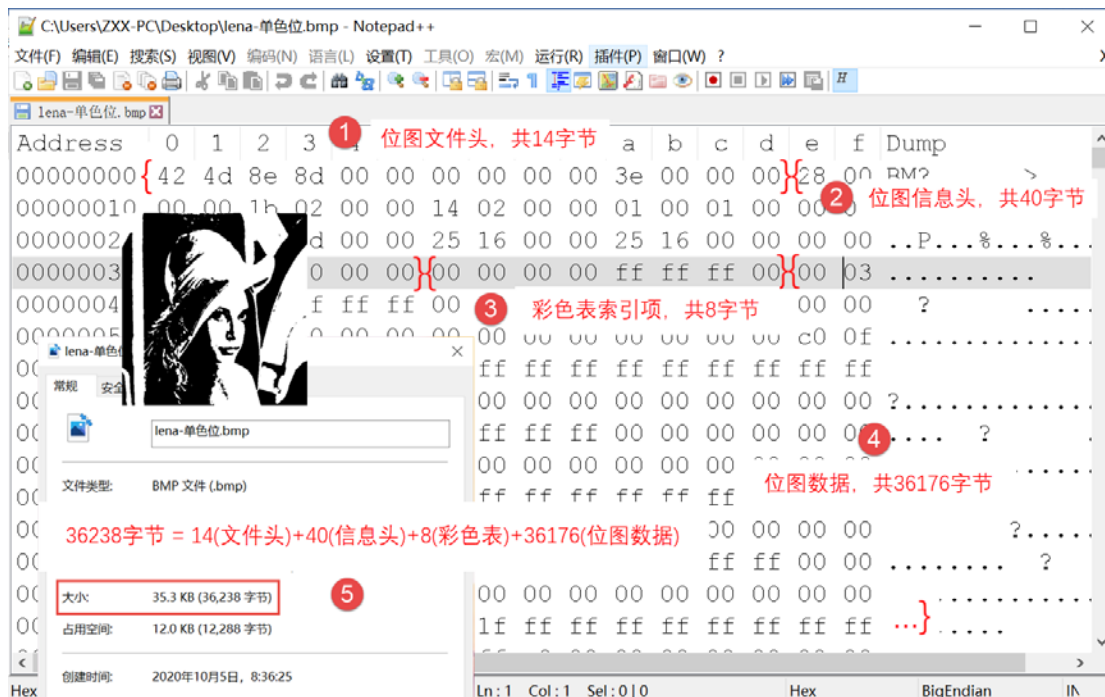


图 3-12

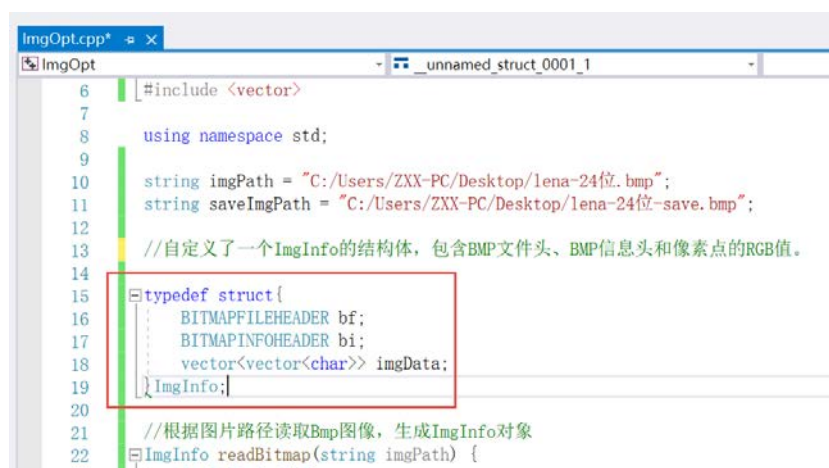
在位图数据存储与读取过程中，有一点需要特别注意：**BMP 像素值的存储规则要求每行的字节数必须是 4 的倍数**。若某行的字节数不是 4 的倍数，需要额外添加字符'0'凑够到 4 的倍数。在对位图数据进行读写时，这一点需要特别留意，否则无法对位图图像进行正确的读写。

4. 使用 C++ 实现对位图文件的读写、显示

在本小节中，将会用 C++ 语言实现对 24 位位图文件的读写以及显示操作。

本程序定义了一个新的结构体 `ImgInfo`，里面包括了位图文件头 `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，加入二维数组 `imgData` 字段的好处是可以使用二维数组更方便地对图像的像素点进行操作。

在程序的主函数中，调用了 `readBitmap`、`showBitmap`、`saveBitmap` 三个函数，分别实现对 BMP 图像的读取、显示、保存操作，如图 4-2 所示。



```
6 | #include <vector>
7 |
8 | using namespace std;
9 |
10 | string imgPath = "C:/Users/ZXX-PC/Desktop/lena-24位.bmp";
11 | string saveImgPath = "C:/Users/ZXX-PC/Desktop/lena-24位-save.bmp";
12 |
13 | //自定义了一个ImgInfo的结构体，包含BMP文件头、BMP信息头和像素点的RGB值。
14 |
15 | typedef struct{
16 |     BITMAPFILEHEADER bf;
17 |     BITMAPINFOHEADER bi;
18 |     vector<vector<char>> imgData;
19 | }ImgInfo;
20 |
21 | //根据图片路径读取Bmp图像，生成ImgInfo对象
22 | ImgInfo readBitmap(string imgPath) {
```

图 4-1



```
109 |     cout << "已保存图像至: " + saveImgPath << endl;
110 | }
111 |
112 | int main() {
113 |     ImgInfo imgInfo = readBitmap(imgPath);
114 |     showBitmap(imgInfo);
115 |     saveBitmap(imgInfo);
116 | }
117 |
```

图 4-2

4.1 位图文件的读取

位图文件读取主要由程序中的 `readBitmap` 函数实现，关键代码如下图 4-3 所示，通过使用 `fread` 函数实现对位图文件头与位图信息头的读取。

```

21 //根据图片路径读取Bmp图像，生成ImgInfo对象
22 ImgInfo readBitmap(string imgPath) {
23     ImgInfo imgInfo;
24     char* buf; //定义文件
25     char* p;
26
27     FILE* fp;
28     fopen_s(&fp, imgPath.c_str(), "rb");
29     if (fp == NULL) {
30         cout << "打开文件失败!" << endl;
31         exit(0);
32     }
33
34     fread(&imgInfo.bf, sizeof(BITMAPFILEHEADER), 1, fp);
35     fread(&imgInfo.bi, sizeof(BITMAPINFOHEADER), 1, fp);
36
37     if (imgInfo.bi.biBitCount != 24) {
38         cout << "不是24位格式的bmp位图!" << endl;
39     }
40 }

```

图 4-3

在下图 4-4 中，通过 fseek 函数与位图文件头的 bfOffBits 字段，对图像像素数据进行定位，以此来读取像素数据信息，并存放到了二维数组中。

注意蓝色框中的代码，由于 BMP 位图采用 4 字节对齐的存储机制，因此可能会存在一些无意义的填充数据，因此我们在读取数据时必须将它们过滤。

```

41
42     fseek(fp, imgInfo.bf.bfOffBits, 0);
43
44     buf = (char*)malloc(imgInfo.bi.biWidth * imgInfo.bi.biHeight * 3);
45     fread(buf, 1, imgInfo.bi.biWidth * imgInfo.bi.biHeight * 3, fp);
46
47     p = buf;
48
49     vector<vector<char>> imgData;
50     for (int y = 0; y < imgInfo.bi.biHeight; y++) {
51         for (int x = 0; x < imgInfo.bi.biWidth; x++) {
52             vector<char> vRGB;
53
54             vRGB.push_back(*(p++)); //blue
55             vRGB.push_back(*(p++)); //green
56             vRGB.push_back(*(p++)); //red
57
58             if (x == imgInfo.bi.biWidth - 1)
59             {
60                 for (int k = 0; k < imgInfo.bi.biWidth % 4; k++) p++;
61             }
62             imgData.push_back(vRGB);
63         }
64     }
65     fclose(fp);

```

图 4-4

4.2 在控制台上显示位图图像

在控制台上显示位图图像，主要由程序中的 showBitmap 函数实现。

由于结构体 ImgInfo 中的 imgData 字段，我们可以很轻易地获取图像的像素值信息，并使用 SetPixel 函数将图像像素显示在控制台特定的位置，这部分的关键代码如下图 4-5 所

示。

需要注意的是，BMP 位图的像素数据存储规则是行内从左到右，行间从下到上。因此，在编程时需要考虑清楚像素点数组与其图像像素实际的坐标位置。

```
void showBitmap(ImgInfo imgInfo) {  
    HWND hWnd;  
    HDC hDc;  
    int yOffset = 150;  
    hWnd = GetForegroundWindow();  
    hDc = GetDC(hWnd);  
  
    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 4-5

4.3 位图文件的保存

位图文件的保存，主要在程序中的 saveBitmap 函数中实现，如下图 4-6 所示。与位图文件的读取类似，按照 BMP 位图的文件结构，先使用 fwrite 函数实现对位图文件头和位图信息头的写入，再遍历像素点将其像素值写入文件中。

同样地，位图的像素数据存取采用 4 字节对齐的方式，当写入的每一行位图数据的长度不足 4 的倍数时，需要填充 '0' 字符。

```
ImgOpt.cpp* -> x  
ImgOpt (全局范围)  
  
88  
89 void saveBitmap(ImgInfo imgInfo) {  
90     FILE* fpw;  
91     fopen_s(&fpw, saveImagePath.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: " + saveImagePath << endl;  
111 }  
112
```

图 4-6

4.4 实验效果

如下图 4-7 所示，在运行程序后，将图像数据读出，然后在控制台上显示图像，最后将图像保存到本地。

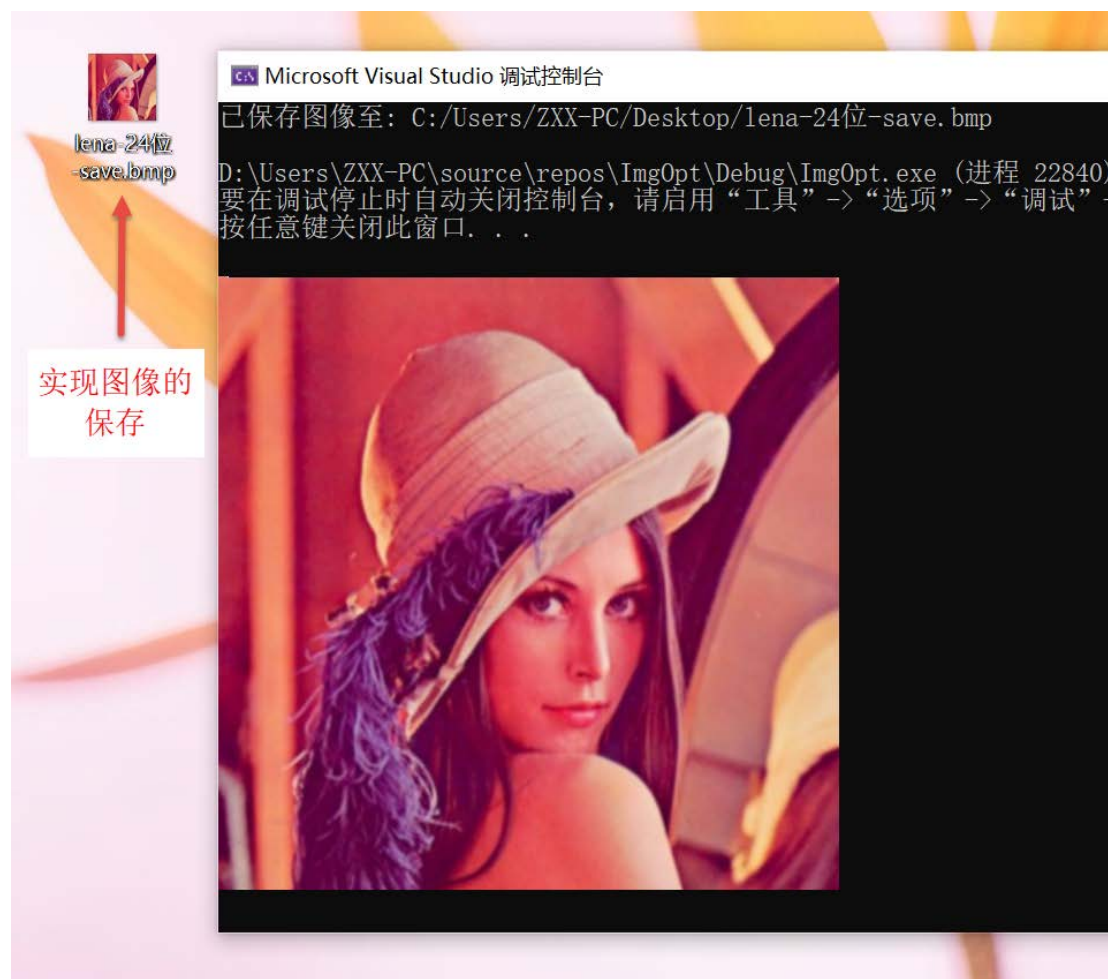


图 4-7

5. 总结

- 位图的文件结构包括四项，分别是：位图文件头、位图信息头、调色板和位图数据。
- 位图文件头存放位图文件的大小、位图数据位置等信息。
- 位图信息头存放位图文件的宽高、图像的位深度、水平/垂直分辨率、位图数据大小等等关键信息。
- 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。
- 位图数据记录了位图的每一个像素的像素值，存储的顺序是行内从左到右，行间从下到上，并要求每行的字节数必须是 4 的倍数。

6. 参考文章

[1] 百度百科--Bitmap 位图.<https://baike.baidu.com/item/Bitmap/6493270?fr=aladdin>

[2] Bitmap 图像格式并用 C++ 读写 Bitmap.
https://blog.csdn.net/weixin_34208185/article/details/86257499

[3] BMP 格式详解.<https://blog.csdn.net/gwwgle/article/details/4775396>

[4] Bitmap 每行 4 字节对齐.https://blog.csdn.net/a_flying_bird/article/details/50585146