**同济大学计算机系**

**高级程序设计大作业报告**

**III**

****

**学 号 1950000**

**姓 名 一二三**

**专 业 计算机类**

**班 级 18级计算机二班**

**授课老师 陈宇飞**

**完成日期 2019/6/3**

文件压缩小程序大作业

一、实验内容

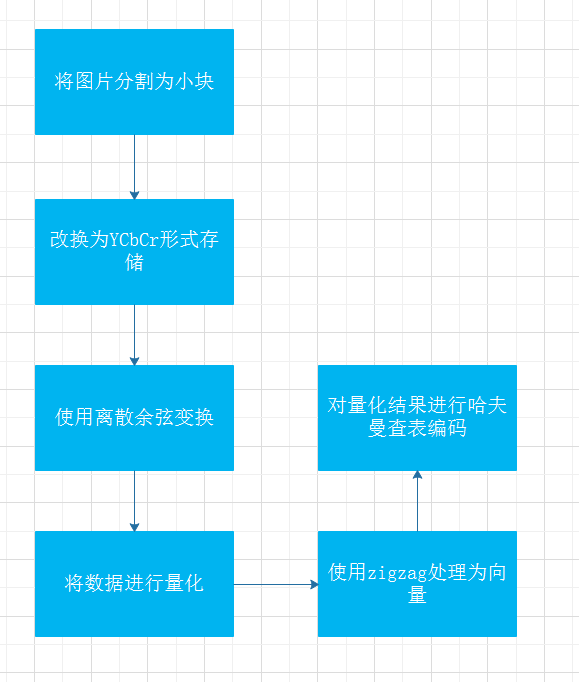
ALPD公司(爱乐普第)名下ALPDOJ(爱乐普第Orange Juice)的销售部小李在系统上尝试添加一个新的产品，但是因为服务器空间寸土寸金，技术员小王限制了产品的示意图只能在500kb，而小李要传的示意图高达3mb，这让他感到很苦恼。于是小李找到小王问能不能提高下文件大小的限制，小王满嘴你知不知道现在硬盘多贵带宽多贵把小李打发走了，这下小李彻底懵了。产品马上就要上市了，网站上可不能没有示意图，走投无路的小李听说你上次为小王压缩了服务器的日志文件，便通过各种渠道又找到了你，希望你能帮帮他压缩一下这些图片，不然消费者看不到示意图销量上不去可是要被炒鱿鱼的。

**二、设计思路与功能描述**

**2.1 项目思路说明**

**2.1.1 压缩算法思路说明**

本次实验的图片压缩算法整体上借鉴了上一周杨闻笛同学的分享内容，即主要基于离散余弦变换与huffman编码来进行压缩，该压缩算法的流程框图如下：



通过上述流程框图不难看出，压缩算法主要由6个部分组成，以下将对这六个环节进行逐一说明。

①将图片分割为小块。由于一般的图片信息处理算法都是针对8×8的像素矩阵来说明的，所以应当将原图片划分为多个8×8的小块，进而方便后续压缩、编码过程中能够逐一对每个小块进行操作。

②将图片更换为YCbCr方式存储。由于对于肉眼来说，对于亮度的敏感性远大于对于色调的敏感性，所以当我们以YCbCr方式存储时，就可以在一定程度上忽略Cb、Cr参数的一些“细节”，而只要保证Y参数的准确性即可。从而可以节约一定的文件大小。

③使用离散余弦变换

离散余弦变换是压缩为jpg图片的核心算法，对于8×8像素矩阵中的每一位置，都可以给出该位置经过离散余弦变换后的结果为：



经过上述离散余弦变换之后，就可以使得图片中较为重要的信息置于F矩阵的左上角，而使得右下角的元素尽可能的靠近于0。从而便于在后续的操作中在F矩阵中得到更多的0，从而达到大幅压缩的目的。

④进行数据量化

由于第③步操作的结果往往是一些小数，而计算机中浮点数的存储往往要占据比整数多得多的空间，不利于压缩率的降低。所以此时应当进行一次数据量化，而数据量化的方式就是让F矩阵中的值与量化系数矩阵Q中对应位置的值相除，并四舍五入取整，从而得到新的矩阵B。

值得一提的是，量化系数矩阵Q的选定是任意的，可以根据实际的开发需要进行选取。但在一般情况下，为了保证亮度信息Y能够尽可能多的保留，亮度信息对应的量化系数矩阵的数值一般较小，借以保留更多的亮度信息；同时为了降低压缩率，可以在一定程度上忽略CbCr信息，所以Cb、Cr对应的量化系数矩阵Q中的元素应当较大一些，使得更多的元素变为0。

而在本次试验中，选取的是推荐的标准亮度量化表与标准色差量化表。

⑤使用zigzag处理为向量

由于信息是存储在计算机中的磁盘上的，而磁盘上数据的存储是线性的，所以在这里应当采用某种方法将量化后的二维矩阵转化为便于计算机存储的向量形式。

而在jpg图片压缩中，常使用的是zigzag方法，该方法就是从（0，0）元素开始，依次斜向绕行，碰到边界后就沿着边界延伸一个单位，再沿着反向的斜线折回，直至遍历完数组中的所有元素。

⑥编码压缩

编码压缩的过程主要分为RLE编码与哈夫曼编码两个步骤。首先应当进行的是RLE编码。

由于在第④步中，发现量化后的矩阵只有左上角有较密集的有效信息，右下角的元素基本上均为0；而该矩阵经过第⑤步的zigzag处理后，正好可以让不为0的元素集中在向量头部，而尾部则填充着大量的0元素。

而RLE算法就是将原本的相量转化为一些列数对，每个数对表示相邻两个不为0的元素之间隔着多少个0。这一编码方式恰好与zigzag处理后向量中含有大量连续0的特点契合，从而可以实现较低的压缩率。

在我们得到这一系列数对后，为了进一步降低压缩率，可以采用哈夫曼编码对这些数对进行编码。不同于第一次大作业中哈夫曼编码的定义，本次试验中的哈夫曼编码更像是一种“打表”行为——即按照已经给定好的哈夫曼表，分别查找每一个数对元素对应的编码即可。

在本次实验中，由于上述两种编码可以在对向量从前向后的遍历中同时平行进行。所以在代码实现中，没有采用杨闻笛同学分享的一次编码的方式，而是采用一次遍历同时编码的方式来实现。

**2.1.2 JPG格式编码的思路说明**

相交于前文中压缩算法的实现，对JPG格式的编码就显得较为简单容易，只需要对照着官方文档的说明依次在对应的位置写入合适的二进制码即可。

按照网上查阅到的官方文档的说明，JPG文件编码的文件头主要包括如下内容：

①SOI（0xFFD8）

表示图像的开始，此时只需要写入0xFFD8即可。

②APP0（0xFFE0）

这里是应用程序的标记，应当留0。此时在写入完APP0后，应该依次写入应用标志长度与标志（“JFIF”）。之后再依次输入主次版本号、单位、水平与竖直分辨率以及水平与竖直点数。

③DQT（0xFFDB）

这里是说明该图片的量化表，在写入完DQT后，首先写入标志长度，在本实验中为132，其次应当逐次以Z型输入本图片使用的量化表。

④SOF0（0xFFC0）

该命令用于表示帧图像开始标记，应当依次写入参数长度、采样精度、行列数、分量数、分量标记、采样因子。使用的量化表以及对应的比例值。

⑤DHT（0xFFC4）

该命令用于定义本张图片所使用的哈夫曼表，在写入完该命令标志与标志长度后，就应当按照高四位表示编码，第四位表示标志的方式进行输入。值得一提的是，高四位中0开头表示DC，1开头表示AC；而第四位中，0开头表示Y，1开头表示C。

⑥SOS（0xFFDA）

该命令表示开始扫描，自此往后，开始向jpg文件写入之前编好码的图片信息，也是图片的主体信息。

⑦EOI（0xFFD9）

该命令表示JPG图片文件写入结束，即作为终结符。

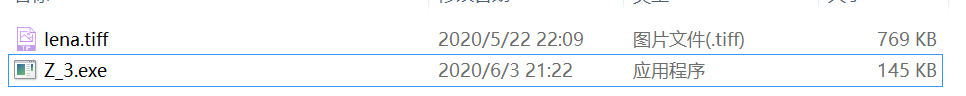
**2.2 功能描述**

**2.2.1 压缩文件生成说明**

将本次实验编码得到的.exe文件与lena.tiff文件置于同一个文件夹中，再通过windows的cmd窗口进入到对应文件夹，即可执行将该文件转化为.jpg文件的操作。

具体流程演示如下：

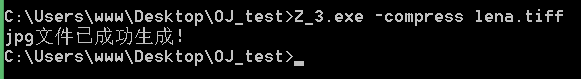
①将.exe文件与lena.tiff文件置于同一个文件夹



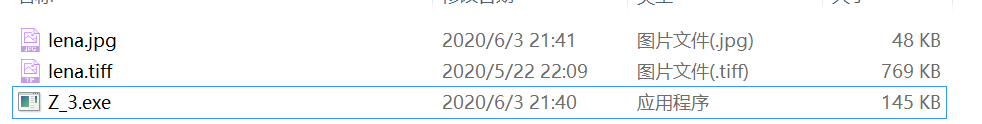
②在cmd窗口使用cd命令进入该目录



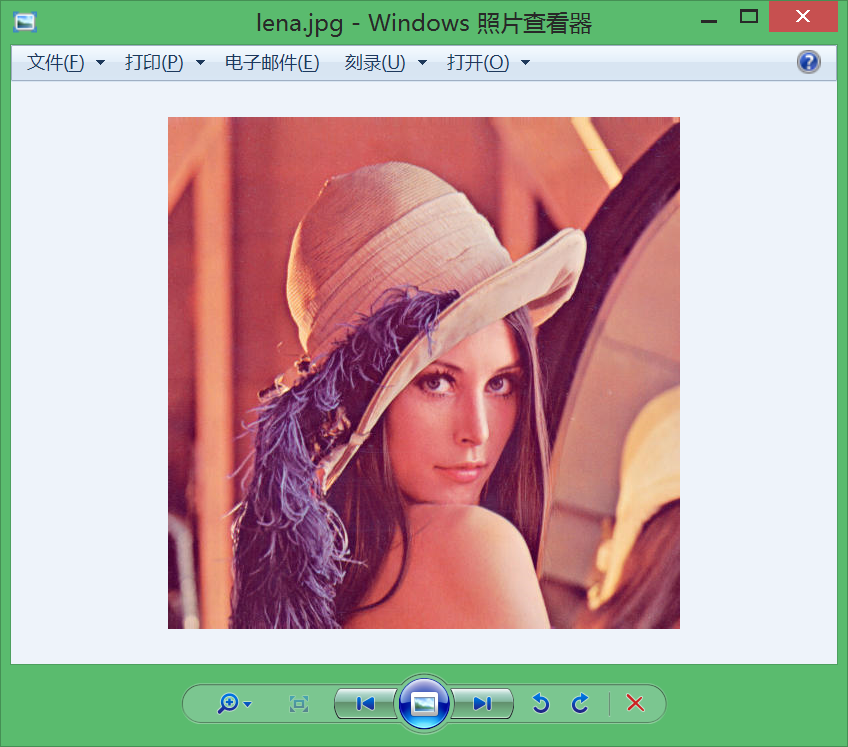
③在cmd窗口输入compress命令



发现弹出提示语，同时在平行目录下生成了对应的lena.jpg



④用windows自带的图片查看器查看图片，发现可以成功打开并显示

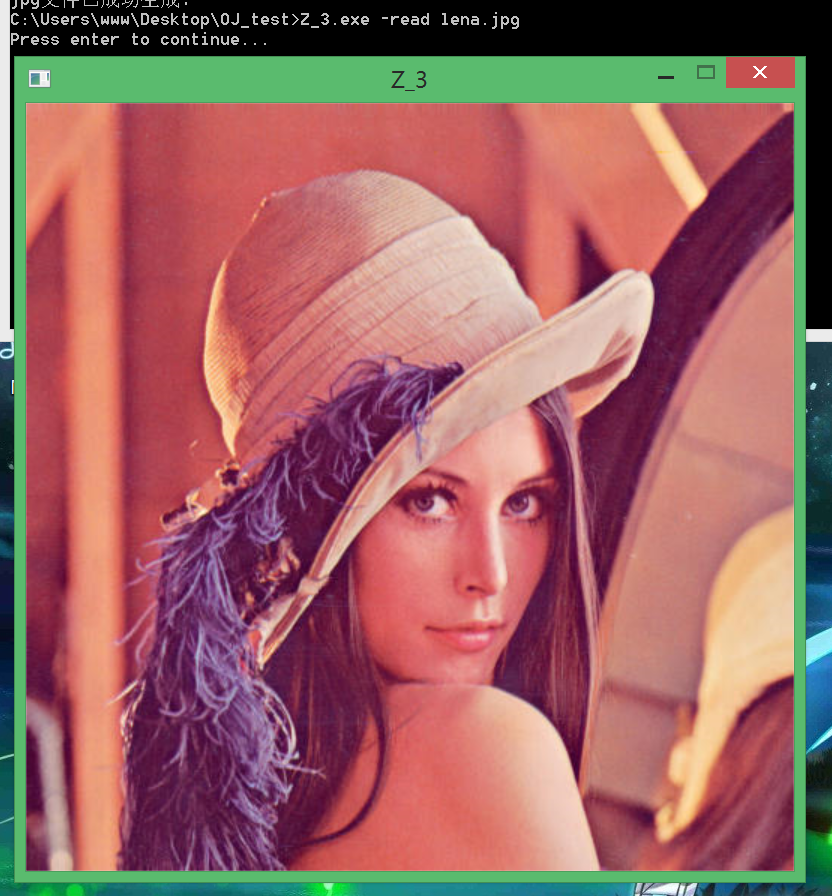


**2.2.2 图片展示命令说明**

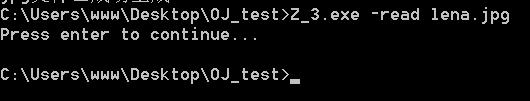
在上述操作的基础上，可继续在cmd窗口中使用-read命令



可以使用easyX自带的框架查看该图片



按下任意键后可退出查看。



三、项目亮点与情况说明

**3.1 项目亮点**

①本次大作业实现了A组方法3与B组方法2，自认为以较高的完成度（90%）完成了本次大作业；

②本次大作业的最终程序Z\_3.exe并没有写死，可以用于转换、查看多种不同规格的jpg文件；

③在编码的部分，采用了一次遍历同时进行RLE编码与哈夫曼编码的方式来替代原有的两次遍历过程；

④以lena.tiff为例，本程序将原本769KB的图片在肉眼可接受的范围内压缩至48KB，压缩比为6.24%，远低于OJ要求的70%。

**3.2 诸多情况说明**

①没有使用分享课上同学给出的DC、AC表

在程序开发的初期，我是按照杨闻笛同学分享课上给出的推荐编码表（https://www.cnblogs.com/buaaxhzh/p/9119870.html）来进行编写的。但在调试的过程中，发现不论如何调整，图片都是乱码，如下图所示：



这一bug我加起来调试了两三天，在重写其他模块依旧无解的情况下，我只好选择放弃推荐编码表重写这一模块。

在最终程序中使用的编码表参考了<https://github.com/AcTarjan/BmpToJpg>中AcTarjan先生的编码表。在对照着AcTarjan先生的代码重写了SOS哈夫曼编码模块后，没有再出现该色块的bug。

经过我个人的思考，我认为出现这种bug的原因可能是我在使用推荐编码表的过程中，对其中的转换方式没有参透，导致出现了纰漏，最终产生bug。

②没有实现B组方法3。

方法3本质上就是jpg图片的解码，该过程本身并不是很难。但正如前文所述，我在哈夫曼编码的过程中debug花费了过长的时间，并多次重写了诸多模块，导致最后并没有余下足够的时间来实现方法3。

但在总体上，我认为本次大作业的完成度较好。

四、遇到的问题与解决方法

**4.1 遇到的问题**

在重写完上述哈夫曼模块后，运行程序可以看到生成的.jpg文件没有了乱码，但色调与亮度明显不合适——我愿称之为“阴间配色”。

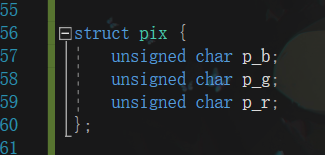


**4.2 解决方法**

解决方法类似于之前大作业一的解决方法，我使用二进制查看器对我自己代码生成的图片与现有程序生成的jpg图片进行逐字节比对，最终发现了问题的根源。

我在编写代码时，只是注意到了jpg文件中每个像素的大小为1个字节，使用了char类型来存储对应值；但却忽略了该类型值不能为负的前提，没有使用unsigned修饰。

这就使得一些本来高亮的色块，因为其亮度值大于127，而“被迫”变为负值，最终使得整个图片看起来较为阴暗。添加了unsigned修饰后的代码为：



在修改代码后，即可生成正确的lena图片。



五、心得体会

六、源代码

#define \_CRT\_SECURE\_NO\_WARNINGS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Root

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "PicReader.h"

#include <stdio.h>

#include <iostream>

#include <math.h>

#include <windows.h>

#include <fstream>

#include <iostream>

using namespace std;

const float PI = 3.1415926f;

const char zigzag[64] = {

0, 1, 5, 6, 14, 15, 27, 28,

2, 4, 7, 13, 16, 26, 29, 42,

3, 8, 12, 17, 25, 30, 41, 43,

9, 11, 18, 24, 31, 40, 44, 53,

10, 19, 23, 32, 39, 45, 52, 54,

20, 22, 33, 38, 46, 51, 55, 60,

21, 34, 37, 47, 50, 56, 59, 61,

35, 36, 48, 49, 57, 58, 62, 63

};

const char Qy[64] = {

16, 11, 10, 16, 24, 40, 51, 61,

12, 12, 14, 19, 26, 58, 60, 55,

14, 13, 16, 24, 40, 57, 69, 56,

14, 17, 22, 29, 51, 87, 80, 62,

18, 22, 37, 56, 68, 109,103,77,

24, 35, 55, 64, 81, 104,113,92,

49, 64, 78, 87, 103,121,120,101,

72, 92, 95, 98, 112,100,103,99

};

const short Qc[64] = {

17, 18, 24, 47, 99, 99, 99, 99,

18, 21, 26, 66, 99, 99, 99, 99,

24, 26, 56, 99, 99, 99, 99, 99,

47, 66, 99, 99, 99, 99, 99, 99,

99, 99, 99, 99, 99, 99, 99, 99,

99, 99, 99, 99, 99, 99, 99, 99,

99, 99, 99, 99, 99, 99, 99, 99,

99, 99, 99, 99, 99, 99, 99, 99

};

const char DCL\_CODES[16] = {

0, 0, 7, 1,

1, 1, 1, 1,

0, 0, 0, 0,

0, 0, 0, 0

};

const unsigned char DCL\_VALUE[12] = {

4, 5, 3, 2,

6, 1, 0, 7,

8, 9, 10, 11

};

const char DCC\_CODES[16] = {

0, 3, 1, 1,

1, 1, 1, 1,

1, 1, 1, 0,

0, 0, 0, 0

};

const unsigned char DCC\_VALUE[12] = {

0, 1, 2, 3,

4, 5, 6, 7,

8, 9, 10, 11

};

const char ACL\_CODES[16] = {

0, 2, 1, 3,

3, 2, 4, 3,

5, 5, 4, 4,

0, 0, 1, 125

};

const unsigned char ACL\_VALUE[162] = {

0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,

0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,

0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,

0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 240,

0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,

0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,

0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,

0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,

0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,

0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,

0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,

0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,

0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,

0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,

0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,

0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,

0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,

0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,

0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,

0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,

0xf9, 0xfa

};

const char ACC\_CODES[16] = {

0, 2, 1, 2,

4, 4, 3, 4,

7, 5, 4, 4,

0, 1, 2, 119

};

const unsigned char ACC\_VALUE[162] = {

0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,

0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,

0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,

0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33, 0x52, 240,

0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34,

0xe1, 0x25, 0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26,

0x27, 0x28, 0x29, 0x2a, 0x35, 0x36, 0x37, 0x38,

0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,

0x49, 0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,

0x59, 0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,

0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,

0x79, 0x7a, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,

0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95, 0x96,

0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5,

0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4,

0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3,

0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2,

0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda,

0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,

0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,

0xf9, 0xfa

};

unsigned short mask[16] = {

1, 2, 4, 8,

16, 32, 64, 128,

256,512,1024,2048,

4096,8192,16384,32768

};

char name\_tiff[100];

char name\_temp[100];

char name\_jpg[100];

struct the\_size {

int height;

int length;

int hin;

int vin;

int value;

};

struct pix {

unsigned char p\_b;

unsigned char p\_g;

unsigned char p\_r;

};

struct pix\_ {

int nY;

int nCb;

int nCr;

};

class photo {

private:

unsigned char QyTable[64], QcTable[64];

the\_size hY\_DC[16], hY\_AC[256], hC\_DC[16], hC\_AC[256];

const char\* fileName;

int size, height, width, \_height, \_width;

short bit\_count;

float temp\_bit[8][8];

the\_size y\_size, cb\_size, cr\_size;

pix\*\* rgb\_board;

pix\_\*\* yc\_board;

void getHuffmanTable(const char\* numTable, const unsigned char\* posTable, the\_size\* huffmanTable) {

int index = 0;

int code = 0;

for (int i = 0; i < 16; i++) {

for (int j = 0; j < numTable[i]; j++) {

huffmanTable[posTable[index]].length = i + 1;

huffmanTable[posTable[index]].value = code;

code++;

index++;

}

code <<= 1;

}

}

int huffmanCoding(int& last, int\* data, the\_size\* HD, the\_size\* HA, the\_size\* output) {

int index = 0;

//DC分量处理

int diff = data[0] - last;

last = data[0];

if (diff == 0) {

output[index++] = HD[0x00];

}

else {

the\_size bs = getValueBit(diff);

output[index++] = HD[bs.length];

output[index++] = bs;

}

//AC分量处理

the\_size EOB = HA[0x00];

the\_size SIXTEENZEROS = HA[240];

int endpos = 63;

while (data[endpos] == 0 && endpos > 0)

endpos--;

for (int i = 1; i <= endpos; i++) {

int startpos = i;

while (data[i] == 0 && i <= endpos)

i++;

int count = i - startpos;

while (count >= 16) {

output[index++] = SIXTEENZEROS;

count -= 16;

}

the\_size bs = getValueBit(data[i]);

output[index++] = HA[count << 4 | bs.length];

output[index++] = bs;

}

if (endpos != 63)

output[index++] = EOB;

return index;

}

void make\_it(int flag, int& x, int& y, int H, int V, int \_H, int \_V, int& last, int& byte, int& bytepos, FILE\* fp) {

int data[64]; the\_size output[128];

int count;

double temp;

float s = 1 / sqrt(8);

for (int i = 0; i < H; i++) {

for (int j = 0; j < V; j++) {

for (int i = x; i < x + 8; i++) {

for (int j = y; j < y + 8; j++) {

if (flag == 1) {

temp = 0.299 \* rgb\_board[i][j].p\_r + 0.587 \* rgb\_board[i][j].p\_g + 0.114 \* rgb\_board[i][j].p\_b;

yc\_board[i][j].nY = round(temp);

}

else if (flag == 2) {

temp = -0.1687 \* rgb\_board[i][j].p\_r - 0.3313 \* rgb\_board[i][j].p\_g + 0.5 \* rgb\_board[i][j].p\_b + 128;

yc\_board[i][j].nCb = round(temp);

}

else if (flag == 3) {

temp = 0.5 \* rgb\_board[i][j].p\_r - 0.4187 \* rgb\_board[i][j].p\_g - 0.0813 \* rgb\_board[i][j].p\_b + 128;

yc\_board[i][j].nCr = round(temp);

}

}

}

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++) {

float sum = 0;

for (int u = 0; u < 8; u++)

for (int v = 0; v < 8; v++) {

if (flag == 1)

sum += temp\_bit[u][i] \* temp\_bit[v][j] \* (yc\_board[x + u][y + v].nY - 128);

else if (flag == 2)

sum += temp\_bit[u][i] \* temp\_bit[v][j] \* (yc\_board[x + u][y + v].nCb - 128);

else if (flag == 3)

sum += temp\_bit[u][i] \* temp\_bit[v][j] \* (yc\_board[x + u][y + v].nCr - 128);

}

float alphaU = i == 0 ? s : 0.5f;

float alphaV = j == 0 ? s : 0.5f;

float temp = alphaU \* alphaV;

if (flag == 1)

data[zigzag[i \* 8 + j]] = round(temp \* sum / QyTable[zigzag[i \* 8 + j]]);

else

data[zigzag[i \* 8 + j]] = round(temp \* sum / QcTable[zigzag[i \* 8 + j]]);

}

if (flag == 1)

count = huffmanCoding(last, data, hY\_DC, hY\_AC, output);

else

count = huffmanCoding(last, data, hC\_DC, hC\_AC, output);

for (int i = 0; i < count; i++) {

int value = output[i].value;

int pos = output[i].length - 1;

while (pos >= 0) {

if (value & mask[pos]) {

byte = byte | mask[bytepos];

}

pos--;

bytepos--;

if (bytepos < 0) {

\_writeByte((unsigned char)byte, fp);

if (byte == 0xff)

\_writeByte((unsigned char)0, fp);

byte = 0;

bytepos = 7;

}

}

}

}

}

y += V \* 8 \* \_V;

if (y >= \_width) {

y = 0;

x += H \* 8 \* \_H;

}

return;

}

the\_size getValueBit(int value) {

int v = value > 0 ? value : -value;

int length = 0;

while (v > 0) {

length++;

v >>= 1;

}

the\_size bs;

bs.length = length;

bs.value = value > 0 ? value : (1 << length) + value - 1;

return bs;

}

//这里定义用于向fp写入内容的函数

void \_writeByte(unsigned char value, FILE\* fp) {

\_write(&value, 1, fp);

}

void \_writeWord(unsigned short value, FILE\* fp) {

unsigned short \_value = ((value >> 8) & 0xFF) | ((value & 0xFF) << 8);

\_write(&\_value, 2, fp);

}

void \_write(const void\* p, int byteSize, FILE\* fp) {

fwrite(p, 1, byteSize, fp);

}

public:

photo(const char\* sFileName = name\_temp) {

fileName = sFileName;

rgb\_board = NULL;

yc\_board = NULL;

FILE\* fp = fopen(name\_temp, "rb");

BITMAPFILEHEADER fh;//文件头

BITMAPINFOHEADER ih;//信息头

fread(&fh, sizeof(BITMAPFILEHEADER), 1, fp);

fread(&ih, sizeof(BITMAPINFOHEADER), 1, fp);

size = fh.bfSize;

height = ih.biHeight;

width = ih.biWidth;

bit\_count = ih.biBitCount;

\_height = (height + 31) / 32 \* 32;

\_width = (width + 31) / 32 \* 32;

rgb\_board = (pix\*\*)malloc(sizeof(pix\*) \* \_height);

for (int i = 0; i < \_height; i++) {

rgb\_board[i] = (pix\*)malloc(sizeof(pix) \* \_width);

}

int offset = ((width \* 3 + 3) / 4 \* 4) - width \* 3;

for (int i = height - 1; i >= 0; i--) {

for (int j = 0; j < width; j++) {

fread(&rgb\_board[i][j], 3, 1, fp);

}

fseek(fp, offset, SEEK\_CUR);

}

fclose(fp);

fp = fopen(name\_temp, "w");

fclose(fp);

}

bool write\_photo() {

int qualityScale = 50;

unsigned char Ythe\_size = 17;

unsigned char Cb\_size = 17;

unsigned char Cr\_size = 17;

FILE\* fp = fopen(name\_jpg, "wb");

if (!fp) {

cout << "false" << endl;

return false;

}

if (qualityScale <= 0) qualityScale = 1;

if (qualityScale >= 100) qualityScale = 99;

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 8; j++) {

int temp = ((int)(Qy[i \* 8 + j] \* qualityScale + 50) / 100);

if (temp <= 0) temp = 1;

if (temp > 0xFF) temp = 0xFF;

QyTable[zigzag[i \* 8 + j]] = (unsigned char)temp;

temp = ((int)(Qc[i \* 8 + j] \* qualityScale + 50) / 100);

if (temp <= 0) temp = 1;

if (temp > 0xFF) temp = 0xFF;

QcTable[zigzag[i \* 8 + j]] = (unsigned char)temp;

}

}

//这里开始写jpg的头

\_writeWord(0xFFD8, fp);

\_writeWord(0xFFE0, fp);

\_writeWord(16, fp);

\_write("JFIF", 5, fp);

\_writeByte(1, fp);

\_writeByte(1, fp);

\_writeByte(1, fp);

\_writeWord(0x78, fp);

\_writeWord(0x78, fp);

\_writeByte(0, fp);

\_writeByte(0, fp);

\_writeWord(0xFFDB, fp);

\_writeWord(132, fp);

\_writeByte(0x00, fp);

\_write(QyTable, 64, fp);

\_writeByte(0x01, fp);

\_write(QcTable, 64, fp);

\_writeWord(0xFFC0, fp);

\_writeWord(17, fp);

\_writeByte(8, fp);

\_writeWord(height & 0xFFFF, fp);

\_writeWord(width & 0xFFFF, fp);

\_writeByte(3, fp);

\_writeByte(1, fp);

\_writeByte(Ythe\_size, fp);

\_writeByte(0, fp);

\_writeByte(2, fp);

\_writeByte(Cb\_size, fp);

\_writeByte(1, fp);

\_writeByte(3, fp);

\_writeByte(Cr\_size, fp);

\_writeByte(1, fp);

\_writeWord(0xFFC4, fp);

\_writeWord(0x01A2, fp);

\_writeByte(0x00, fp);

\_write(DCL\_CODES, sizeof(DCL\_CODES), fp);

\_write(DCL\_VALUE, sizeof(DCL\_VALUE), fp);

\_writeByte(0x10, fp);

\_write(ACL\_CODES, sizeof(ACL\_CODES), fp);

\_write(ACL\_VALUE, sizeof(ACL\_VALUE), fp);

\_writeByte(0x01, fp);

\_write(DCC\_CODES, sizeof(DCC\_CODES), fp);

\_write(DCC\_VALUE, sizeof(DCC\_VALUE), fp);

\_writeByte(0x11, fp);

\_write(ACC\_CODES, sizeof(ACC\_CODES), fp);

\_write(ACC\_VALUE, sizeof(ACC\_VALUE), fp);

\_writeWord(0xFFDA, fp);

\_writeWord(12, fp);

\_writeByte(3, fp);

\_writeByte(1, fp);

\_writeByte(0x00, fp);

\_writeByte(2, fp);

\_writeByte(0x11, fp);

\_writeByte(3, fp);

\_writeByte(0x11, fp);

\_writeByte(0, fp);

\_writeByte(0x3F, fp);

\_writeByte(0, fp);

//接下来开始写图片的存储内容

int maxH, maxV;

int lastY = 0, lastCb = 0, lastCr = 0;

int byte = 0, bytepos = 7;

y\_size.height = (Ythe\_size & 240) >> 4;

cb\_size.height = (Cb\_size & 240) >> 4;

cr\_size.height = (Cr\_size & 240) >> 4;

y\_size.length = Ythe\_size & 15;

cb\_size.length = Cb\_size & 15;

cr\_size.length = Cr\_size & 15;

maxH = max(max(y\_size.height, cb\_size.height), cr\_size.height);

maxV = max(max(y\_size.length, cb\_size.length), cr\_size.length);

y\_size.vin = maxV / y\_size.length;

cb\_size.vin = maxV / cb\_size.length;

cr\_size.vin = maxV / cr\_size.length;

y\_size.hin = maxH / y\_size.height;

cb\_size.hin = maxH / cb\_size.height;

cr\_size.hin = maxH / cr\_size.height;

\_height = (height + maxH \* 8 - 1) / (maxH \* 8) \* maxH \* 8;

\_width = (width + maxV \* 8 - 1) / (maxV \* 8) \* maxV \* 8;

//为yc\_board分配内存

yc\_board = (pix\_\*\*)malloc(sizeof(pix\_\*) \* \_height);

for (int i = 0; i < \_height; i++)

yc\_board[i] = (pix\_\*)malloc(sizeof(pix\_) \* \_width);

//初始化temp\_bit

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 8; j++) {

temp\_bit[i][j] = cos((2 \* i + 1) \* j \* PI / 16);

}

}

//以下初始化哈夫曼表

memset(hY\_DC, 0, sizeof(hY\_DC));

getHuffmanTable(DCL\_CODES, DCL\_VALUE, hY\_DC);

memset(hY\_AC, 0, sizeof(hY\_AC));

getHuffmanTable(ACL\_CODES, ACL\_VALUE, hY\_AC);

memset(hC\_DC, 0, sizeof(hC\_DC));

getHuffmanTable(DCC\_CODES, DCC\_VALUE, hC\_DC);

memset(hC\_AC, 0, sizeof(hC\_AC));

getHuffmanTable(ACC\_CODES, ACC\_VALUE, hC\_AC);

int x1 = 0, y1 = 0, x2 = 0, y2 = 0, x3 = 0, y3 = 0;

while (1) {

make\_it(1, x1, y1, y\_size.height, y\_size.length, y\_size.hin, y\_size.vin, lastY, byte, bytepos, fp);

make\_it(2, x2, y2, cb\_size.height, cb\_size.length, cb\_size.hin, cb\_size.vin, lastCb, byte, bytepos, fp);

make\_it(3, x3, y3, cr\_size.height, cr\_size.length, cr\_size.hin, cr\_size.vin, lastCr, byte, bytepos, fp);

if (x1 >= \_height)

break;

}

//最后写入结束标志

\_writeWord(0xFFD9, fp);

fclose(fp);

remove(name\_temp);

return true;

}

};

void Write\_temp(BYTE\* img, int h, int w)

{

int l = (w \* 3 + 3) / 4 \* 4;

int bmi[] = { l \* h + 54,0,54,40,w,h,1950000,0,l \* h,0,0,100,0 };

FILE\* fp;

int err = fopen\_s(&fp, name\_temp, "wb");

if (!fp) {

printf("%s", "error!");

exit(-1);

}

fprintf\_s(fp, "BM");//标识“BM”

fwrite(&bmi, 52, 1, fp);

fwrite(img, 1, h\*l, fp);

fclose(fp);

}

void readLena() {

PicReader imread;

BYTE\* data = nullptr;

UINT x, y;

imread.readPic(name\_tiff);

imread.getData(data, x, y);

//imread.showPic(data, x, y);

BYTE\* get\_pix = new BYTE[(size\_t)x \* y \* 3];

int index = 0;

for (int i = x - 1; i >= 0; i--) {

for (int j = 0; j < y \* 4; j += 4) {

get\_pix[index++] = data[i\*y \* 4 + j + 2];

get\_pix[index++] = data[i\*y \* 4 + j + 1];

get\_pix[index++] = data[i\*y \* 4 + j];

}

}

Write\_temp(get\_pix, x, y);

delete[] data;

data = nullptr;

}

void compress() {

readLena();

photo bmp;

bmp.write\_photo();

cout << "jpg文件已成功生成！";

}

void read() {

PicReader imread;

BYTE\* data = nullptr;

UINT x, y;

imread.readPic(name\_jpg);

imread.getData(data, x, y);

imread.showPic(data, x, y);

}

int main(int argc, char\* argv[]) {

char temp[100];

if (argc != 3) {

return -1;

}

if (strcmp(argv[1], "-compress") == 0) {

strcpy(name\_tiff, argv[2]);

int i = 0;

for (i = 0; name\_tiff[i] != '.'; i++) {

name\_temp[i] = name\_tiff[i];

name\_jpg[i] = name\_tiff[i];

}

name\_temp[i] = '.';

name\_temp[i + 1] = 't';

name\_temp[i + 2] = 'e';

name\_temp[i + 3] = 'm';

name\_temp[i + 4] = 'p';

name\_temp[i + 5] = '\0';

name\_jpg[i] = '.';

name\_jpg[i + 1] = 'j';

name\_jpg[i + 2] = 'p';

name\_jpg[i + 3] = 'g';

name\_jpg[i + 4] = '\0';

compress();

}

else if (strcmp(argv[1], "-read") == 0) {

strcpy(name\_jpg, argv[2]);

read();

}

else {

return -1;

}

return 0;

}