**图像压缩小程序**

**课程名称：面向对象程序设计**

**姓名：张宇**

**院系：电子与信息工程学院计算机科学与技术专业**

**班级：计算机科学与技术1班**

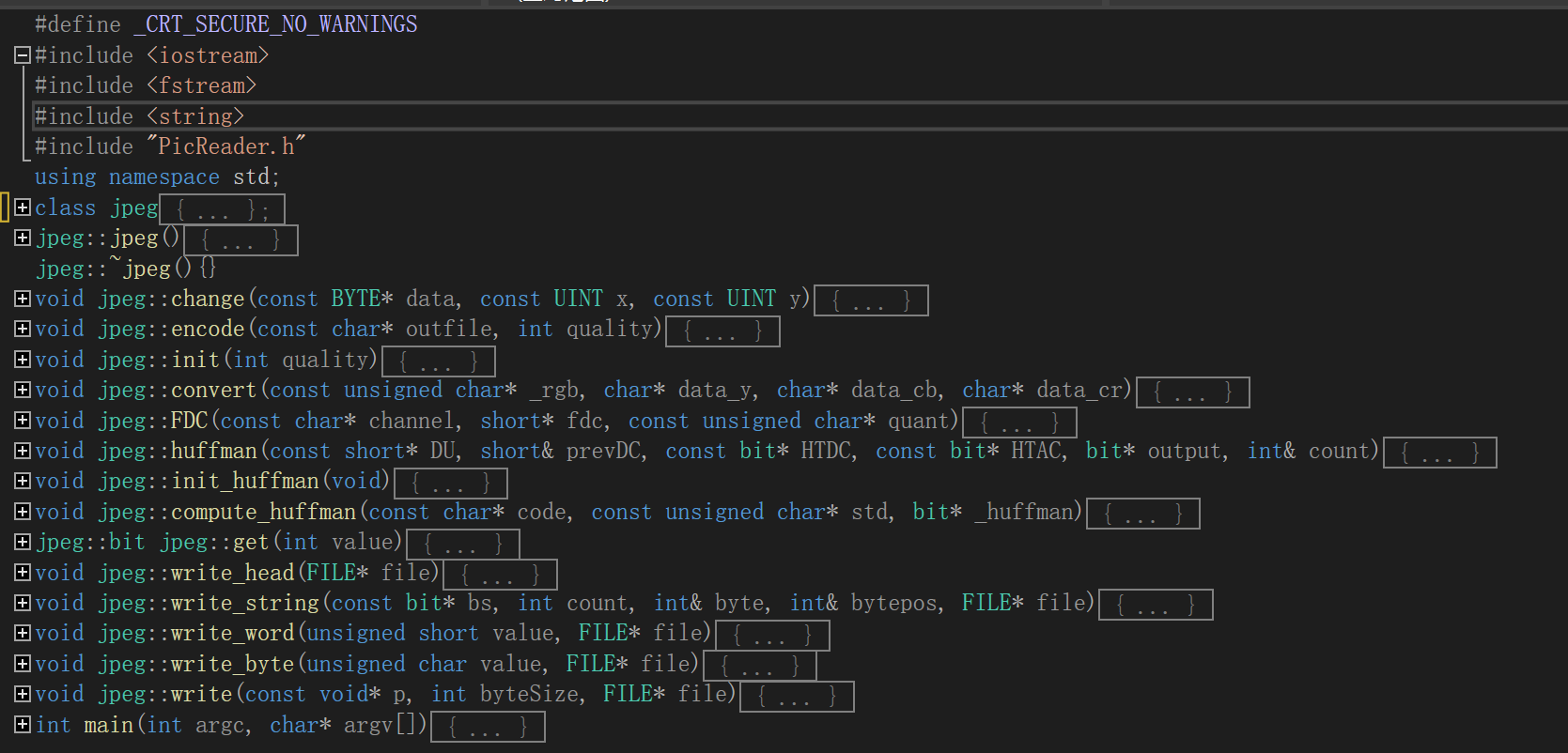
**学号：2251745**

**完成日期：2023年12月12日**

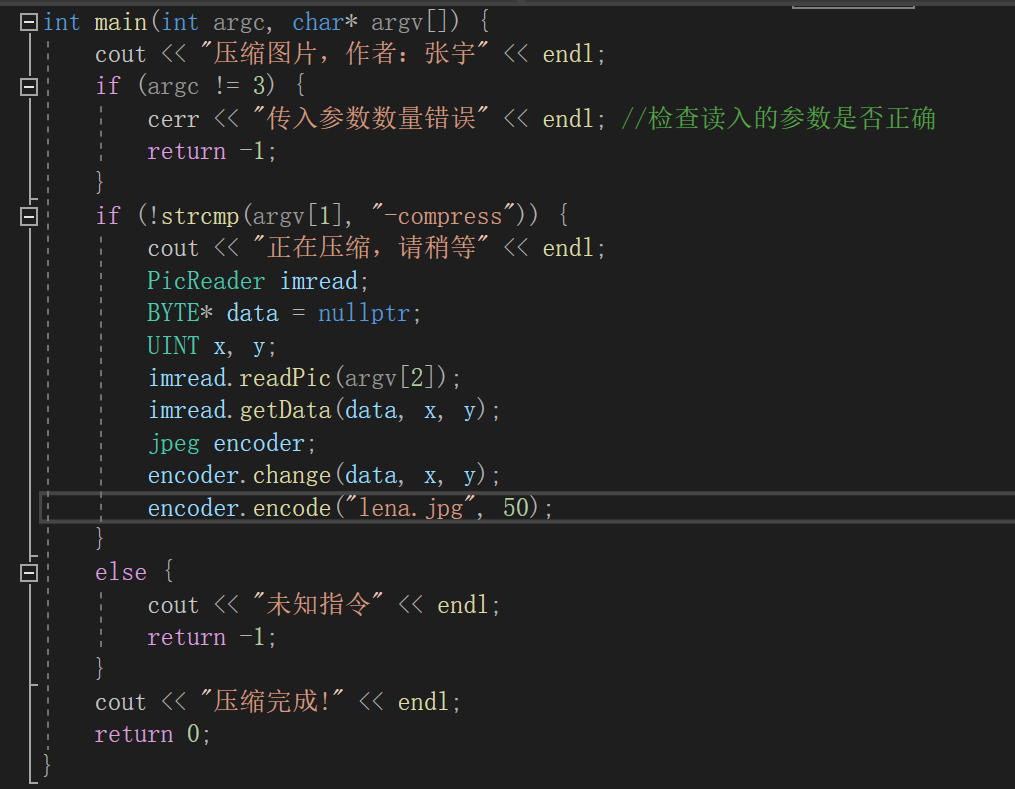
**1、设计思路与功能描述**

在刚拿到这个题目的时候还是处于一个比较懵的状态，毕竟图像跟文件还是有一定区别的，一开始没什么思路，然后看了资料中的demo程序后才有点明白，读取图像后也还是一种编码形式，也可以像文件压缩那样对其压缩。但毕竟是刚开始做，肯定还是要有点梦想的，所以在大概明白了底层逻辑后还是想尝试一下频域编码算法。于是就开始了漫长的找资料了解的过程，在大致了解了一下算法是如何实现的以后，选择先实现特殊格式编码，然后再考虑要不要解码的问题，从.tiff文件编码成.jpg文件又是一个全新的问题，又是漫漫长路。毕竟想要编码成.jpg文件需要大量格式化的头数据等等，这就需要从网上查找资料，以及了解如何对数据进行转化并输出，从网上的资料再到自己手搓代码，这又是一个很需要时间的过程，写完代码后已耗时良久，在看看解码，又是毫无头绪，我在编码时使用了有损压缩，所以对于一个有损压缩的文件要如何解码我一头雾水，也就作罢了。

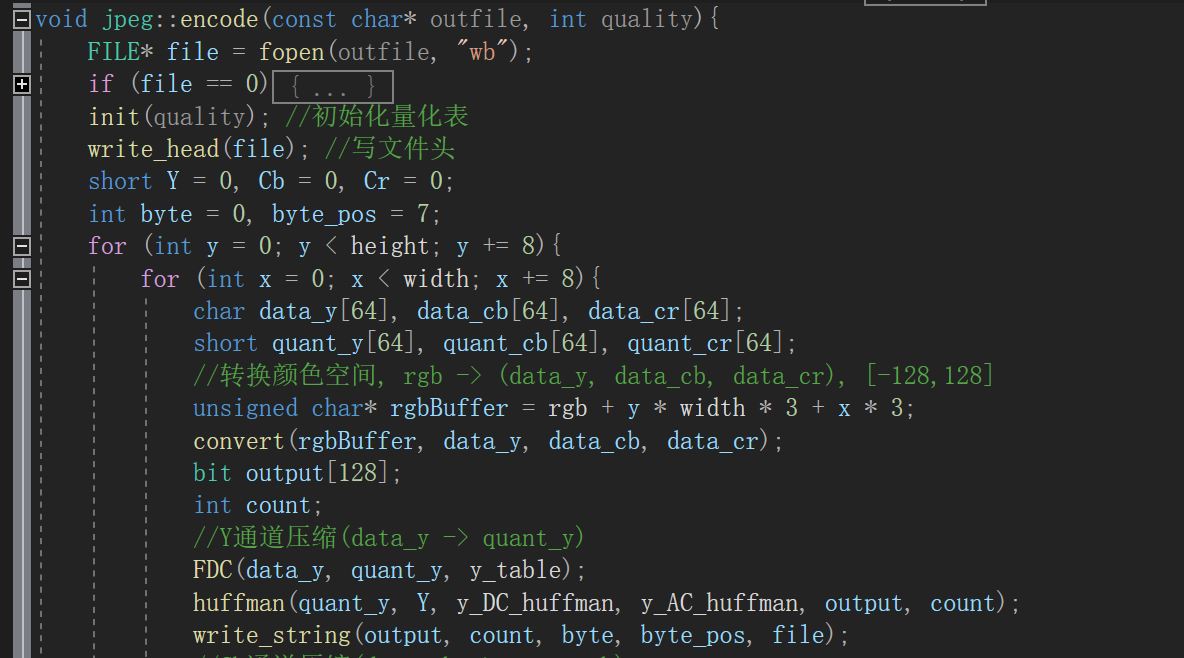
这是我本题用到的函数总览。

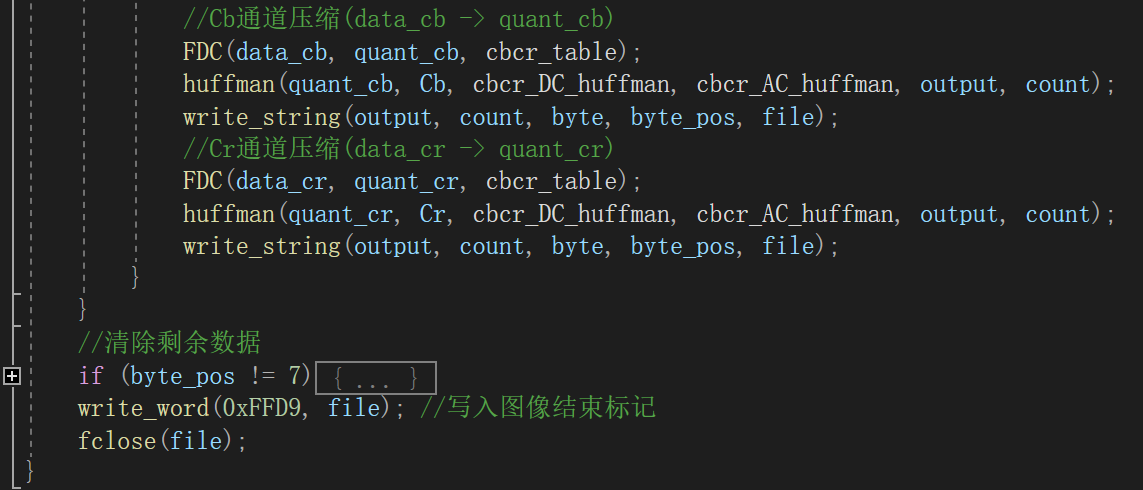


这是我本题中的main函数内容，使用cmdline处理输入文件，实现了将lena.tiff文件压缩并写成能使用 Windows 默认的图片查看器成功打开的lena.jpg格式的文件。在主函数中，我先调用提供的库函数从lena.tiff文件中读入数据，然后使用change函数将读入数据中的不透明度信息去除，再使用encode函数完成压缩并编码到lena.jpg文件中。



这是我本次程序中最重要的encode编码函数了，在本函数中，我先打开了一个待写入文件，然后对量化表进行初始化，接着就是将.jpg文件所特有的文件头进行写入操作，再将整个图像进行分割，并对每个部分进行颜色空间转换操作，将RGB颜色空间转换到YCbCr颜色空间，然后分别对Y通道、Cb通道和Cr通道进行压缩，最后清除剩余数据，再写入图像结束标记，关闭文本，完成本函数的所有内容。为了完成上述提到的各种功能，在本函数中还需要调用上文函数总览中提到过的各式各样的函数，为了节省篇幅，这里就不对这些函数进行一一展示，在本报告的第4部分的源代码展示中可以展开每个函数的功能以及具体实现方法。





**2、在实验过程中遇到的问题与解决方法**

（1）问题：在一开始对于如何编码成.jpg文件是一头雾水的，不知道应该从何下手。

解决方法：遇到不会的知识肯定还是首选网上冲浪，在经过了漫长的资料查询后大致有了一定的了解，然后开始着手构建代码的框架，并一步步地填充具体函数内容。对于.jpg文件所特有的格式问题当然还是需要借鉴一下前辈们的经验，除此之外，还有色度、亮度量化表，标准直流\交流亮度\色度哈夫曼编码表等知识。

（2）问题：在写完程序进行调试时，发现lena的颜色发生了改变。

解决方法：对简单纯色块进行压缩，查看压缩结果后发现红蓝色块的压缩颜色正好相反，而其他颜色色块的压缩结果没有问题，然后想到应该是在转换RGB颜色空间到YCbCr颜色空间时发生错误，一开始我仔细检查了公式的系数，没有发现问题，然后才看到是R、G、B的值读成了B、G、R，才出现颜色相反的问题。

（3）问题：在将数据压缩并编码成.jpg文件后不知道该如何进行解码的操作。

解决方法：我所使用的压缩编码方式是有损压缩，再加上对于编码的理解比较局限，所以对于如何解码一头雾水，并没有得到解决。

**3、心得体会**

本次作业实现的图像压缩功能所使用到的基本都是没有学过的知识，比上次的文件压缩功能是更加的陌生，例如频域编码算法、将文件写成特殊格式等等，这就需要我们各种途径查找相关的资料，学习相关的知识，并且能够在理解掌握之后在自己的代码中实现相应的功能，这就很考验我们的理解和学习能力了，同时这也是最需要时间的工作。因此，如何快速、高效地寻找到合适的资料学习，就很需要我们的检索能力了。我也是在这上面消耗了大量的时间，但其实效果也并不是很好，只能说是越看感觉自己越乱，所以还需要值得注意的是在感觉自己理解掌握了一点后，就应该及时地敲敲代码，自己动手实践一下，“纸上得来终觉浅，须知此事要躬行”，只有当自己敲代码的时候才知道到底哪里掌握得还不够好，也才可以相对应地去学习。

写了很多次大作业后感觉到其实每次大作业看着都很唬人，尤其是没有循序渐进的过程时，总觉得自己好像做不出来，掌握的知识不够解决问题，但其实只有自己动起手来，一点一点地完成代码，才能知道自己的不足，也才有学习的方向和动力，一个个小功能的实现，一个个函数的编码，最终叠加起来，原来一个看似不可能的大作业也就一步步的解决了。

**4、源代码**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <string>

#include "PicReader.h"

using namespace std;

class jpeg {

private:

int width;// 图像宽度

int height;// 图像高度

unsigned char\* rgb;// RGB 像素数据

unsigned char y\_table[64];// 亮度量化表

unsigned char cbcr\_table[64];// 色度量化表

// 亮度量化表

const unsigned char luminance[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

};

// 色度量化表

const unsigned char chrominance[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

};

// zigzag扫描顺序表

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

};

// 标准直流亮度哈夫曼编码表

const char DC\_luminance\_NRcode[16] = { 0, 0, 7, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 };

const unsigned char DC\_luminance\_value[12] = { 4, 5, 3, 2, 6, 1, 0, 7, 8, 9, 10, 11 };

// 标准直流色度哈夫曼编码表

const char DC\_chrominance\_NRcode[16] = { 0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 };

const unsigned char DC\_chrominance\_value[12] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

// 标准交流亮度哈夫曼编码表

const char AC\_luminance\_NRcode[16] = { 0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d };

const unsigned char AC\_luminance\_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, 0xf0,

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

};

// 标准交流色度哈夫曼编码表

const char AC\_chrominance\_NRcode[16] = { 0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77 };

const unsigned char AC\_chrominance\_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, 0xf0,

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

};

struct bit {

int length;// 编码位数

int value;// 编码值

};

bit y\_DC\_huffman[12]; // 亮度DC哈夫曼表

bit y\_AC\_huffman[256];// 亮度AC哈夫曼表

bit cbcr\_DC\_huffman[12];// 色度DC哈夫曼表

bit cbcr\_AC\_huffman[256];// 色度AC哈夫曼表

void init(int quality);

void init\_huffman(void);

void compute\_huffman(const char\* code, const unsigned char\* std, bit\* \_huffman);

void convert(const unsigned char\* \_rgb, char\* data\_y, char\* data\_cb, char\* data\_cr);

void FDC(const char\* channel, short\* fdc, const unsigned char\* quant);

void huffman(const short\* DU, short& prevDC, const bit\* HTDC, const bit\* HTAC, bit\* output, int& count);

bit get(int value);

void write\_head(FILE\* file);

void write\_string(const bit\* bs, int count, int& byte, int& bytepos, FILE\* file);

void write\_word(unsigned short value, FILE\* file);

void write\_byte(unsigned char value, FILE\* file);

void write(const void\* p, int byteSize, FILE\* file);

public:

jpeg();

~jpeg();

void change(const BYTE\* data, const UINT x, const UINT y);

void encode(const char\* outfile, int quality);

};

jpeg::jpeg() : width(0), height(0), rgb(0) {

//初始化静态表格

init\_huffman();

}

jpeg::~jpeg() {}

void jpeg::change(const BYTE\* data, const UINT x, const UINT y) {

int size = x \* y \* 3;

unsigned char\* buffer = new(nothrow)unsigned char[size];

if (buffer == NULL) {

cout << "内存分配失败" << endl;

exit(1);

}

for (DWORD i = 0; i < x \* y \* 3; i += 3) {

buffer[i] = data[i / 3 \* 4];

buffer[i + 1] = data[i / 3 \* 4 + 1];

buffer[i + 2] = data[i / 3 \* 4 + 2];

}

rgb = buffer;

width = x;

height = y;

}

void jpeg::encode(const char\* outfile, int quality) {

FILE\* file = fopen(outfile, "wb");

if (file == 0) {

cout << "文件打开失败" << endl;

exit(1);

}

init(quality); //初始化量化表

write\_head(file); //写文件头

short Y = 0, Cb = 0, Cr = 0;

int byte = 0, byte\_pos = 7;

for (int y = 0; y < height; y += 8) {

for (int x = 0; x < width; x += 8) {

char data\_y[64], data\_cb[64], data\_cr[64];

short quant\_y[64], quant\_cb[64], quant\_cr[64];

//转换颜色空间, rgb -> (data\_y, data\_cb, data\_cr), [-128,128]

unsigned char\* rgbBuffer = rgb + y \* width \* 3 + x \* 3;

convert(rgbBuffer, data\_y, data\_cb, data\_cr);

bit output[128];

int count;

//Y通道压缩(data\_y -> quant\_y)

FDC(data\_y, quant\_y, y\_table);

huffman(quant\_y, Y, y\_DC\_huffman, y\_AC\_huffman, output, count);

write\_string(output, count, byte, byte\_pos, file);

//Cb通道压缩(data\_cb -> quant\_cb)

FDC(data\_cb, quant\_cb, cbcr\_table);

huffman(quant\_cb, Cb, cbcr\_DC\_huffman, cbcr\_AC\_huffman, output, count);

write\_string(output, count, byte, byte\_pos, file);

//Cr通道压缩(data\_cr -> quant\_cr)

FDC(data\_cr, quant\_cr, cbcr\_table);

huffman(quant\_cr, Cr, cbcr\_DC\_huffman, cbcr\_AC\_huffman, output, count);

write\_string(output, count, byte, byte\_pos, file);

}

}

//清除剩余数据

if (byte\_pos != 7) {

write\_byte(byte, file);

}

write\_word(0xFFD9, file); //写入图像结束标记

fclose(file);

}

void jpeg::init(int quality) {

// 限制quality的取值范围在(1, 99)

if (quality <= 0) {

quality = 1;

}

if (quality >= 100) {

quality = 99;

}

// 根据quality初始化亮度和色度的量化表

for (int i = 0; i < 64; i++) {

// 初始化亮度量化表

int temp = ((int)(luminance[i] \* quality + 50) / 100);

if (temp <= 0) {

temp = 1;

}

if (temp > 0xFF) {

temp = 0xFF;

}

y\_table[zigzag[i]] = (unsigned char)temp;

// 初始化色度量化表

temp = ((int)(chrominance[i] \* quality + 50) / 100);

if (temp <= 0) {

temp = 1;

}

if (temp > 0xFF) {

temp = 0xFF;

}

cbcr\_table[zigzag[i]] = (unsigned char)temp;

}

}

void jpeg::convert(const unsigned char\* \_rgb, char\* data\_y, char\* data\_cb, char\* data\_cr) {

// 转换RGB颜色空间到YCbCr颜色空间

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

const unsigned char\* p = \_rgb + y \* width \* 3;

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

unsigned char R = \*p++;

unsigned char G = \*p++;

unsigned char B = \*p++;

data\_y[y \* 8 + x] = (char)(0.29871 \* R + 0.58661 \* G + 0.11448 \* B - 128);

data\_cb[y \* 8 + x] = (char)(-0.16874 \* R - 0.33126 \* G + 0.50000 \* B);

data\_cr[y \* 8 + x] = (char)(0.50000 \* R - 0.41869 \* G - 0.08131 \* B);

}

}

}

void jpeg::FDC(const char\* channel, short\* fdc, const unsigned char\* quant) {

const float PI = 3.1415926f;

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

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

float \_u = (float)((u == 0) ? 1 / sqrtf(8.0) : 0.5);

float \_v = (float)((v == 0) ? 1 / sqrtf(8.0) : 0.5);

float temp = 0;

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

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

float data = channel[y \* 8 + x];

data \*= cosf((2 \* x + 1) \* u \* PI / 16);

data \*= cosf((2 \* y + 1) \* v \* PI / 16);

temp += data;

}

}

int index = zigzag[v \* 8 + u];

//量化

temp \*= \_u \* \_v / quant[index];

fdc[index] = (short)((short)(temp + 16384.5) - 16384);

}

}

}

void jpeg::huffman(const short\* DU, short& prevDC, const bit\* HTDC, const bit\* HTAC, bit\* output, int& count) {

bit EOB = HTAC[0x00];

bit SIXTEEN\_ZEROS = HTAC[0xF0];

int index = 0;

// 编码DC分量

int dc = (int)(DU[0] - prevDC);

prevDC = DU[0];

if (dc == 0) {

output[index++] = HTDC[0];

}

else {

bit bs = get(dc);

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

output[index++] = bs;

}

// 编码AC分量

int end = 63;

while ((end > 0) && (DU[end] == 0)) {

end--;

}

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

int start = i;

while ((DU[i] == 0) && (i <= end)) i++;

int zero = i - start;

if (zero >= 16) {

for (int j = 1; j <= zero / 16; j++)

output[index++] = SIXTEEN\_ZEROS;

zero = zero % 16;

}

bit bs = get(DU[i]);

output[index++] = HTAC[(zero << 4) | bs.length];

output[index++] = bs;

}

if (end != 63) {

output[index++] = EOB;

}

count = index;

}

void jpeg::init\_huffman(void) {

// 初始化亮度DC哈夫曼表

memset(&y\_DC\_huffman, 0, sizeof(y\_DC\_huffman));

compute\_huffman(DC\_luminance\_NRcode, DC\_luminance\_value, y\_DC\_huffman);

// 初始化亮度AC哈夫曼表

memset(&y\_AC\_huffman, 0, sizeof(y\_AC\_huffman));

compute\_huffman(AC\_luminance\_NRcode, AC\_luminance\_value, y\_AC\_huffman);

// 初始化色度DC哈夫曼表

memset(&cbcr\_DC\_huffman, 0, sizeof(cbcr\_DC\_huffman));

compute\_huffman(DC\_chrominance\_NRcode, DC\_chrominance\_value, cbcr\_DC\_huffman);

// 初始化色度AC哈夫曼表

memset(&cbcr\_AC\_huffman, 0, sizeof(cbcr\_AC\_huffman));

compute\_huffman(AC\_chrominance\_NRcode, AC\_chrominance\_value, cbcr\_AC\_huffman);

}

void jpeg::compute\_huffman(const char\* code, const unsigned char\* std, bit\* \_huffman) {

unsigned char pos = 0;

unsigned short code\_value = 0;

// 根据标准哈夫曼表和码字长度计数构建自定义哈夫曼表

for (int k = 1; k <= 16; k++) {

for (int j = 1; j <= code[k - 1]; j++) {

\_huffman[std[pos]].value = code\_value;

\_huffman[std[pos]].length = k;

pos++;

code\_value++;

}

code\_value <<= 1;

}

}

jpeg::bit jpeg::get(int value) {

bit ret;

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

// 计算bit的长度

int length = 0;

for (length = 0; v; v >>= 1) {

length++;

}

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

ret.length = length;

return ret;

};

void jpeg::write\_head(FILE\* file) {

//SOI

write\_word(0xFFD8, file); // marker = 0xFFD8

//APPO

write\_word(0xFFE0, file); // marker = 0xFFE0

write\_word(16, file); // length = 16 for usual JPEG, no thumbnail

write("JFIF", 5, file); // 'JFIF\0'

write\_byte(1, file); // version\_hi

write\_byte(1, file); // version\_low

write\_byte(0, file); // xyunits = 0 no units, normal density

write\_word(1, file); // xdensity

write\_word(1, file); // ydensity

write\_byte(0, file); // thumbWidth

write\_byte(0, file); // thumbHeight

//DQT

write\_word(0xFFDB, file); //marker = 0xFFDB

write\_word(132, file); //size=132

write\_byte(0, file); //QTYinfo== 0: bit 0..3: number of QT = 0 (table for Y)

// bit 4..7: precision of QT

// bit 8 : 0

write(y\_table, 64, file); //YTable

write\_byte(1, file); //QTCbinfo = 1 (quantization table for Cb,Cr)

write(cbcr\_table, 64, file); //CbCrTable

//SOFO

write\_word(0xFFC0, file); //marker = 0xFFC0

write\_word(17, file); //length = 17 for a truecolor YCbCr JPG

write\_byte(8, file); //precision = 8: 8 bits/sample

write\_word(height & 0xFFFF, file); //height

write\_word(width & 0xFFFF, file); //width

write\_byte(3, file); //nrofcomponents = 3: We encode a truecolor JPG

write\_byte(1, file); //IdY = 1

write\_byte(0x11, file); //HVY sampling factors for Y (bit 0-3 vert., 4-7 hor.)(SubSamp 1x1)

write\_byte(0, file); //QTY Quantization Table number for Y = 0

write\_byte(2, file); //IdCb = 2

write\_byte(0x11, file); //HVCb = 0x11(SubSamp 1x1)

write\_byte(1, file); //QTCb = 1

write\_byte(3, file); //IdCr = 3

write\_byte(0x11, file); //HVCr = 0x11 (SubSamp 1x1)

write\_byte(1, file); //QTCr Normally equal to QTCb = 1

//DHT

write\_word(0xFFC4, file); //marker = 0xFFC4

write\_word(0x01A2, file); //length = 0x01A2

write\_byte(0, file); //HTYDCinfo bit 0..3 : number of HT (0..3), for Y =0

// bit 4 : type of HT, 0 = DC table,1 = AC table

// bit 5..7 : not used, must be 0

write(DC\_luminance\_NRcode, sizeof(DC\_luminance\_NRcode), file); //DC\_L\_NRC

write(DC\_luminance\_value, sizeof(DC\_luminance\_value), file); //DC\_L\_VALUE

write\_byte(0x10, file); //HTYACinfo

write(AC\_luminance\_NRcode, sizeof(AC\_luminance\_NRcode), file);

write(AC\_luminance\_value, sizeof(AC\_luminance\_value), file); //we'll use the standard Huffman tables

write\_byte(0x01, file); //HTCbDCinfo

write(DC\_chrominance\_NRcode, sizeof(DC\_chrominance\_NRcode), file);

write(DC\_chrominance\_value, sizeof(DC\_chrominance\_value), file);

write\_byte(0x11, file); //HTCbACinfo

write(AC\_chrominance\_NRcode, sizeof(AC\_chrominance\_NRcode), file);

write(AC\_chrominance\_value, sizeof(AC\_chrominance\_value), file);

//SOS

write\_word(0xFFDA, file); //marker = 0xFFC4

write\_word(12, file); //length = 12

write\_byte(3, file); //nrofcomponents, Should be 3: truecolor JPG

write\_byte(1, file); //Idy=1

write\_byte(0, file); //HTY bits 0..3: AC table (0..3)

// bits 4..7: DC table (0..3)

write\_byte(2, file); //IdCb

write\_byte(0x11, file); //HTCb

write\_byte(3, file); //IdCr

write\_byte(0x11, file); //HTCr

write\_byte(0, file); //Ss not interesting, they should be 0,63,0

write\_byte(0x3F, file); //Se

write\_byte(0, file); //Bf

}

void jpeg::write\_string(const bit\* bs, int count, int& byte, int& bytepos, FILE\* file) {

// 位掩码数组，用于逐位检查码字值

unsigned short mask[] = { 1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768 };

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

int value = bs[i].value;

int len = bs[i].length - 1;

while (len >= 0)

{

if ((value & mask[len]) != 0)

{

byte = byte | mask[bytepos];

}

len--;

bytepos--;

if (bytepos < 0) {

// 写入流

write\_byte((unsigned char)(byte), file);

if (byte == 0xFF) {

// 处理特殊情况

write\_byte((unsigned char)(0x00), file);

}

// 重新初始化

bytepos = 7;

byte = 0;

}

}

}

}

void jpeg::write\_word(unsigned short value, FILE\* file) {

// 将16位数据按字节顺序写入文件

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

write(&\_value, 2, file);

}

void jpeg::write\_byte(unsigned char value, FILE\* file) {

// 写入一个字节的数据到文件

write(&value, 1, file);

}

void jpeg::write(const void\* p, int byteSize, FILE\* file) {

// 将指定字节大小的数据写入文件

fwrite(p, 1, byteSize, file);

}

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

cout << "压缩图片，作者：张宇" << endl;

if (argc != 3) {

cerr << "传入参数数量错误" << endl; //检查读入的参数是否正确

return -1;

}

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

cout << "正在压缩，请稍等" << endl;

PicReader imread;

BYTE\* data = nullptr;

UINT x, y;

imread.readPic(argv[2]);

imread.getData(data, x, y);

jpeg encoder;

encoder.change(data, x, y);

encoder.encode("lena.jpg", 50);

cout << "压缩完成!" << endl;

}

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

PicReader imread;

BYTE\* data = nullptr;

UINT x, y;

imread.readPic(argv[2]);

imread.getData(data, x, y);

imread.showPic(data, x, y);

}

else {

cout << "未知指令" << endl;

return -1;

}

return 0;

}