同濟大學
TONGJI UNIVERSITY

高级语言程序设计大作业

装

订

线

目录

装 订 线

1. 设计思路与功能描述

1.1 设计思路：

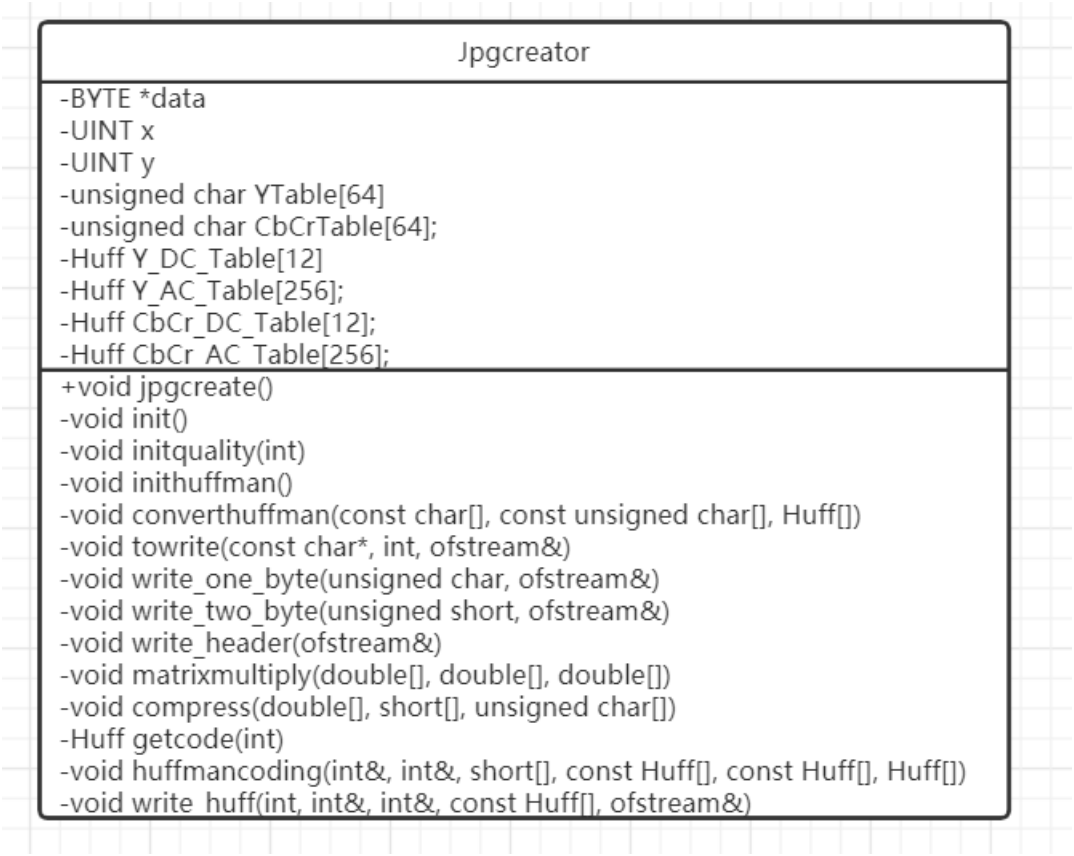
本次大作业核心在于利用频域压缩算法实现对图像的压缩，同时将图像压缩实现编码到 JPG 格式并且能使用 Windows 默认的图片查看器成功打开文件。

大作业使用面向对象编程，根据实现要求设计出 Jpgcreator 类。

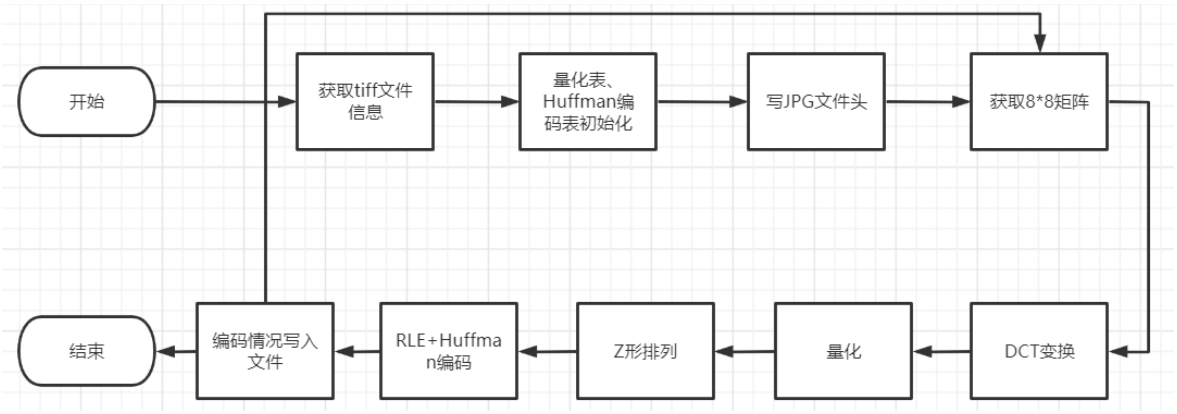
类属性：读入的数据 data、读入数据矩阵长 x、读入数据矩阵高 y、2 个量化表 YTable、CbCrTable、4 个 Huffman 编码表 Y_DC_Table、Y_AC_Table、CbCr_DC_Table、CbCr_DC_Table。

类方法：总处理中心 jpgcreate、初始化 init、量化表初始化 initquaity、Huffman 编码表初始化 inithuffman、转化 Huffman 编码表 converthuffman、写入文件 towrite、写 1 个字节 write_ont_byte、写 2 个字节 write_two_byte、写入文件头 write_header、DCT 变换 martixmultiply、量化和 Z 形排列 compress、获取编码 getcode、进行 Huffman 编码 huffmancoding、编码结果写入文件 write_huff。

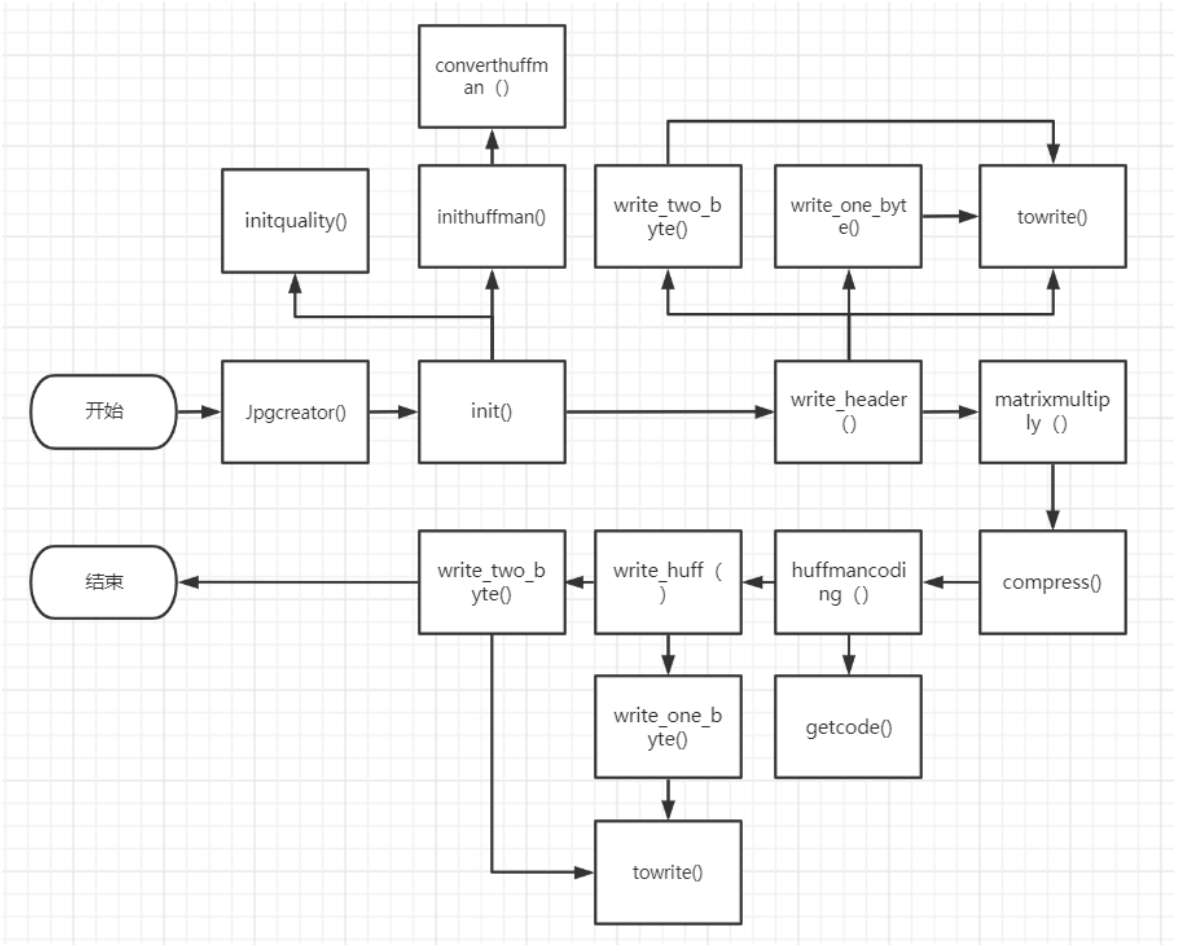
类图如下：



Jpg 生成过程如下：



类的方法调用流程图如下：



1.2 功能描述：

Jpgcreator 类:

Jpgcreator(char[]):

构造函数，获取 data、x、y。

~Jpgcreator():

析构函数，释放 data

void jpgcreate():
总处理中心，调用其他方法完成压缩和生成 jpg 文件。
void init():
初始化，调用 initquality()和 inithuffman()。
void initquality(int):
量化表初始化
void inithuffman():
huffman 编码表初始化
void converthuffman(const char[], const unsigned char[], Huff[]):
转换编码表
void towrite(const char*, int, ofstream&):
写入文件
void write_one_byte(unsigned char, ofstream&):
写一个字节
void write_two_byte(unsigned short, ofstream&):
写两个字节
void write_header(ofstream&):
写入文件头
void matrixmultiply(double[], double[], double[]):
DCT 变换
void compress(double[], short[], unsigned char[]):
量化+Z 排
Huff getcode(int):
获取编码
void huffmancoding(int&, int&, short[], const Huff[], const Huff[], Huff[]):
RLE+huffman 编码
void write_huff(int, int&, int&, const Huff[], ofstream&):
编码结果写入文件

2.在实验过程中遇到的问题及解决方法

2.1 概念理解：

如何理解傅里叶相关变换？

通过查阅资料，以及 http://www.jezzamon.com/fourier/zh-cn.html 这个网站的科普，理解了傅里叶变换的概念，以及在图像压缩中的应用，进而了解到各类傅里叶相关变换。厘清了傅里叶变换的核心在于把一堆数据转换为易于压缩的另一堆数据。

2.2JPG 压缩过程：

2.2.1 如何选择合适的量化表、Huffman 编码表？

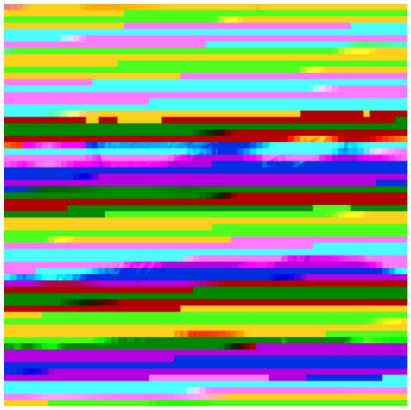
通过查阅相关资料，获取 JPG 官方推荐的相关表，并在程序中使用。

2.2.2 如何平衡压缩比和压缩质量？

通过一个 ratio 参数，调整标准量化表，从而改变处理后数据中 0 的个数，影响 RLE，影响

最终的 JPG 文件。以此调整压缩比和图片质量，对两者进行平衡。

2.2.3 图片出现下面的奇怪线条？



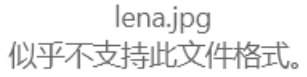
原因是 DC 分量没有采用差分编码。通过应用上差分编码，解决了上述问题。

2.3 编写 JPG 文件

2.3.1 如何写文件头？

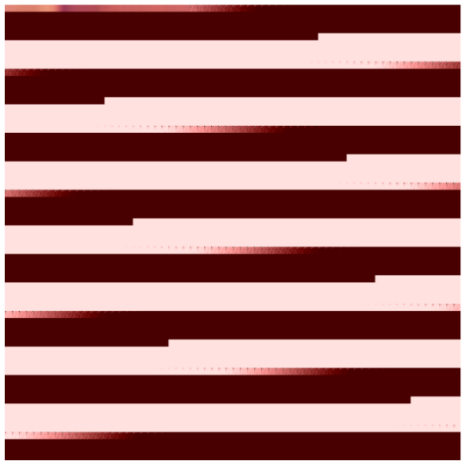
利用 Notepad++的 Hex Editor 插件，查看 jpg 文件头具体信息，从而掌握 jpg 文件头的内容，进而自己可以写文件头。

2.3.2 出现下面的文件格式错误



原因是没有将写入的两个字节的顺序调整。通过与、或操作将高低位转换，再写入文件，解决了上述问题。

2.3.3 为何图片出现奇怪线条？



原因是没有对 0XFF 特殊处理。如果需要写入 0XFF 字节，需要在写入 0XFF 字节之后再写

入填充字节 0X00。这是由于 JPEG 编码器默认会将 0XFF 作为提示字段处理。为了表明这真的是一个数据段而不是提示符 ，再写进去一个 0X00 作为填充字节即可。

3. 心得体会

3.1 程序和算法设计的体会：

1）压缩技术领域博大精深，内容庞杂。各类无损方法、有损方法、有损无损结合方法层出不穷，很值得研究。

1）对于图像、音频这些信息，我们一般采用有损压缩，关键在于选择合适的压缩算法，同时也要注意压缩比和图像质量的平衡。

2）频域编码很神奇，而且在许多领域都会出现，对于这样陌生知识的掌握，需要抛开细枝末节，直接掌握其本质和核心，并能快速应用。

3）程序设计要注重完备性、鲁棒性。对于程序要实现的功能，尽量写完整，写全面。对变量的处理，对指针的使用和回收都要注意。

4）对于矩阵乘法和傅里叶变换，还可以通过分治的方法，进行进一步的速度优化。例如 Strassen 矩阵乘法和快速傅里叶变换等都可以提高速度，如果在遇到大的矩阵处理时可以使用。

5）JPG、PNG 等各类文件都有自己的编码方法，编码中的标识内容是编码成功的关键，可以用各种方式查看具体的编码内容，从而 DEBUG。

3.2 编程过程的体会：

1）编程过程中要善于 Debug。程序中出现 bug 非常正常，通过断点、单步调试、变量监控等往往能快速锁定 bug。

2）一定要注意编码细节，比如 8*8 矩阵的选取、比如写入 2 个字节的顺序，比如 DC 分量的差分编码等，图片出现问题，往往是在细节上出现问题。

3）实践出真知。通过亲自实验、反复尝试，就能找到相对较好的程序和算法设计。

4）搜索引擎是自学过程中的好老师。在遇到自己不了解的部分时，可以去 cppreference、微软 c++文档、csdn、stackoverflow 等地方寻找解决方法。

4. 源代码：

```
#define _CRT_SECURE_NO_WARNINGS
#include"PicReader.h"
#include<cmath>
#include<fstream>
#include<iostream>
using namespace std;
//DCT矩阵
double quotient[64] =
{
    0.353553, 0.353553, 0.353553, 0.353553, 0.353553, 0.353553, 0.353553, 0.353553,
    0.490393, 0.415735, 0.277785, 0.0975452, -0.0975451, -0.277785, -0.415735, -0.490393,
    0.46194, 0.191342, -0.191342, -0.46194, -0.46194, -0.191342, 0.191342, 0.46194,
    0.415735, -0.0975451, -0.490393, -0.277785, 0.277785, 0.490393, 0.0975453, -0.415735,
    0.353553, -0.353553, -0.353553, 0.353553, 0.353554, -0.353553, -0.353554, 0.353553,
    0.277785, -0.490393, 0.097545, 0.415735, -0.415735, -0.0975454, 0.490393, -0.277785,
    0.191342, -0.46194, 0.46194, -0.191342, -0.191342, 0.46194, -0.46194, 0.191341,
```

```
    0.0975452, -0.277785, 0.415735, -0.490393, 0.490393, -0.415735, 0.277785, -0.0975447
};
//DCT转置矩阵
double quotientT[64] =
{
    0.353553, 0.490393, 0.46194, 0.415735, 0.353553, 0.277785, 0.191342, 0.0975452,
    0.353553, 0.415735, 0.191342, -0.0975451, -0.353553, -0.490393, -0.46194, -0.277785,
    0.353553, 0.277785, -0.191342, -0.490393, -0.353553, 0.097545, 0.46194, 0.415735,
    0.353553, 0.0975452, -0.46194, -0.277785, 0.353553, 0.415735, -0.191342, -0.490393,
    0.353553, -0.0975451, -0.46194, 0.277785, 0.353554, -0.415735, -0.191342, 0.490393,
    0.353553, -0.277785, -0.191342, 0.490393, -0.353553, -0.0975454, 0.46194, -0.415735,
    0.353553, -0.415735, 0.191342, 0.0975453, -0.353554, 0.490393, -0.46194, 0.277785,
    0.353553, -0.490393, 0.46194, -0.415735, 0.353553, -0.277785, 0.191341, -0.0975447
};
//标准亮度量化表
const unsigned char standard_YTable[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
};
//标准色度量化表
const unsigned char standard_CbCrTable[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
};
//Z形排列表
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
};
//亮度直流huffman编码表
const char standard_Y_DC_Code[] = { 0, 0, 7, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 };
const unsigned char standard_Y_DC_Value[] = { 4, 5, 3, 2, 6, 1, 0, 7, 8, 9, 10, 11 };
//亮度交流huffman编码表
```

```
const char standard_Y_AC_Code[] = { 0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d };
const unsigned char standard_Y_AC_Value[] =
{
    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
};
//色度直流huffman编码表
const char standard_CbCr_DC_Code[] = { 0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 };
const unsigned char standard_CbCr_DC_Value[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
//色度交流huffman编码表
const char standard_CbCr_AC_Code[] = { 0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77 };
const unsigned char standard_CbCr_AC_Value[] =
{
    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
};
```

装 订 线

```
//数据最高位
unsigned short highest[] = { 1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768 };
//范式huffman编码对应长度和值
struct Huff
{
    int length;
    int value;
};
//类
class Jpgcreator
{
private:
    BYTE *data;//读入的数据
    UINT x;//数据矩阵长
    UINT y;//矩阵高
    unsigned char YTable[64];//量化表
    unsigned char CbCrTable[64];
    Huff Y_DC_Table[12];//huffman编码表
    Huff Y_AC_Table[256];
    Huff CbCr_DC_Table[12];
    Huff CbCr_AC_Table[256];
public:
    void jpgcreate();//总处理中心
    Jpgcreator(char[]);
    ~Jpgcreator();
private:
    void init();//初始化
    void initquality(int);//量化表初始化
    void inithuffman();//huffman编码表初始化
    void converthuffman(const char[], const unsigned char[], Huff[]);//转换编码表
    void towrite(const char*, int, ofstream&);//写入文件
    void write_one_byte(unsigned char, ofstream&);//写一个byte
    void write_two_byte(unsigned short, ofstream&);//写两个byte
    void write_header(ofstream&);//写入文件头
    void matrixmultiply(double[], double[], double[]);//DCT变换
    void compress(double[], short[], unsigned char[]);//量化+Z排
    Huff getcode(int);//获取编码
    void huffmancoding(int&, int&, short[], const Huff[], const Huff[], Huff[]);//huffman编
码
    void write_huff(int, int&, int&, const Huff[], ofstream&);//编码结果写入文件
};
//构造函数
Jpgcreator::Jpgcreator(char file[])
{
    PicReader imread;
    imread.readPic(file);
    imread.getData(data, x, y);
    memset(YTable, 0, sizeof(YTable));
    memset(CbCrTable, 0, sizeof(CbCrTable));
    memset(Y_DC_Table, 0, sizeof(Y_DC_Table));
    memset(Y_AC_Table, 0, sizeof(Y_AC_Table));
    memset(CbCr_DC_Table, 0, sizeof(CbCr_DC_Table));
    memset(CbCr_AC_Table, 0, sizeof(CbCr_AC_Table));
}
```

装 订 线

```
//析构函数
Jpgcreator::~Jpgcreator()
{
    delete[] data;
    data = nullptr;
}
//量化表初始化
void Jpgcreator::initquality(int ratio)
{
    for (int i = 0; i < 64; i++)
    {
        int temp = (int)(standard_YTable[i] * ratio / 100);
        if (temp <= 0)
            temp = 1;
        YTable[i] = (unsigned char)temp;

        temp = (int)(standard_CbCrTable[i] * ratio / 100);
        if (temp <= 0)
            temp = 1;
        CbCrTable[i] = (unsigned char)temp;
    }
}
//转换编码表
void Jpgcreator::converthuffman(const char Code[], const unsigned char Value[], Huff
Table[])
{
    int pos = 0;
    int value = 0;
    for (int i = 0; i < 16; i++)
    {
        for (int j = 0; j < Code[i]; j++)
        {
            Table[Value[pos]].value = value;
            Table[Value[pos]].length = i + 1;
            pos++;
            value++;
        }
        value <<= 1;
    }
}
//huffman编码表初始化
void Jpgcreator::inithuffman()
{
    converthuffman(standard_Y_DC_Code, standard_Y_DC_Value, Y_DC_Table);//标准表转化为实际编
码表

    converthuffman(standard_Y_AC_Code, standard_Y_AC_Value, Y_AC_Table);

    converthuffman(standard_CbCr_DC_Code, standard_CbCr_DC_Value, CbCr_DC_Table);

    converthuffman(standard_CbCr_AC_Code, standard_CbCr_AC_Value, CbCr_AC_Table);
}
//初始化
void Jpgcreator::init()
```

```
{
    cout << "注意：压缩率越大，压缩文件越小，但会导致图片质量下降\n";
    cout << "请输入压缩率(建议1~100之间)：\n";
    int ratio;
    cin >> ratio;
    initquality(ratio);
    inithuffman();
}
//写入文件
void Jpgcreator::towrite(const char* p, int size, ofstream& fout)
{
    fout.write(p, size);
}
//写一个byte
void Jpgcreator::write_one_byte(unsigned char value, ofstream& fout)
{
    towrite((char*)&value, 1, fout);
}
//写两个byte
void Jpgcreator::write_two_byte(unsigned short value, ofstream& fout)
{
    unsigned short value1 = ((value >> 8) & 0xFF) | ((value & 0xFF) << 8);//高低位转换
    towrite((char*)&value1, 2, fout);
}
//写入文件头
void Jpgcreator::write_header(ofstream& fout)
{
    //SOI
    write_two_byte(0xFFD8, fout);//图像开始
    //APP0
    write_two_byte(0xFFE0, fout);//应用程序保留标记0
    write_two_byte(16, fout);//长度
    towrite("JFIF", 5, fout);//标识符
    write_one_byte(1, fout);//版本号
    write_one_byte(1, fout);
    write_one_byte(0, fout);//密度单位
    write_two_byte(1, fout);//像素密度
    write_two_byte(1, fout);
    write_one_byte(0, fout);//像素数目
    write_one_byte(0, fout);
    //DQT
    write_two_byte(0xFFDB, fout);//定义量化表
    write_two_byte(132, fout);//长度
    write_one_byte(0, fout);//量化表ID
    towrite((char*)YTable, 64, fout);//亮度
    write_one_byte(1, fout);//ID
    towrite((char*)CbCrTable, 64, fout);//色度
    //SOF0
    write_two_byte(0xFFC0, fout);//帧图像开始
    write_two_byte(17, fout);//长度
    write_one_byte(8, fout);//精度
    write_two_byte(y & 0xFFFF, fout);//高度
    write_two_byte(x & 0xFFFF, fout);//宽度
```

```
    write_one_byte(3, fout);//分量数
    write_one_byte(1, fout);//颜色分量信息1
    write_one_byte(0x11, fout);
    write_one_byte(0, fout);
    write_one_byte(2, fout);//颜色分量信息2
    write_one_byte(0x11, fout);
    write_one_byte(1, fout);
    write_one_byte(3, fout);//颜色分量信息3
    write_one_byte(0x11, fout);
    write_one_byte(1, fout);
    //DHT
    write_two_byte(0xFFC4, fout);//定义哈夫曼表
    write_two_byte(0x01A2, fout);//长度
    write_one_byte(0, fout);//表ID
    towrite(standard_Y_DC_Code, sizeof(standard_Y_DC_Code), fout);//不同位数的码字数量
    towrite((char*)standard_Y_DC_Value, sizeof(standard_Y_DC_Value), fout);//编码内容（下同
理）
    write_one_byte(0x10, fout);
    towrite(standard_Y_AC_Code, sizeof(standard_Y_AC_Code), fout);
    towrite((char*)standard_Y_AC_Value, sizeof(standard_Y_AC_Value), fout);
    write_one_byte(0x01, fout);
    towrite(standard_CbCr_DC_Code, sizeof(standard_CbCr_DC_Code), fout);
    towrite((char*)standard_CbCr_DC_Value, sizeof(standard_CbCr_DC_Value), fout);
    write_one_byte(0x11, fout);
    towrite(standard_CbCr_AC_Code, sizeof(standard_CbCr_AC_Code), fout);
    towrite((char*)standard_CbCr_AC_Value, sizeof(standard_CbCr_AC_Value), fout);
    //SOS
    write_two_byte(0xFFDA, fout);//扫描开始
    write_two_byte(12, fout);//长度
    write_one_byte(3, fout);//分量数
    write_one_byte(1, fout);//颜色分量信息1
    write_one_byte(0, fout);
    write_one_byte(2, fout);//颜色分量信息2
    write_one_byte(0x11, fout);
    write_one_byte(3, fout);//颜色分量信息3
    write_one_byte(0x11, fout);
    write_one_byte(0, fout);//压缩图像数据
    write_one_byte(0x3F, fout);
    write_one_byte(0, fout);
}
//DCT变换
void Jpgcreator::matrixmultiply(double A[], double B[], double result[])
{
    //矩阵乘法
    double t = 0;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            t = 0;
            for (int k = 0; k < 8; k++)
                t += A[i*8+k] * B[k*8+j];
            result[i*8+j] = t;
        }
```

```
    }
}
//量化+Z排
void Jpgcreator::compress(double raw_data[], short final_data[], unsigned char table[])
{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            double temp = raw_data[i * 8 + j] / table[zigzag[i * 8 + j]];
            final_data[zigzag[i * 8 + j]] = (short)(temp);
        }
    }
}
//获取编码
Huff Jpgcreator::getcode(int value)
{
    Huff temp;
    int length = 0;
    int value_fabs = (value > 0) ? value : -value;
    for (length = 0; value_fabs; value_fabs >>= 1)//获取长度
        length++;
    temp.value = value > 0 ? value : ((1 << length) + value - 1);//获取值
    temp.length = length;
    return temp;
};
//RLE+huffman编码
void Jpgcreator::huffmancoding(int& form, int& Length, short data[], const Huff DC[], const
Huff AC[], Huff Out[])
{
    Huff EOB = AC[0x00];//后续都为零
    Huff SIXTEEN_ZEROS = AC[0xF0];//连续16零
    int length = 0;//初始化长度
    int last = 63;//AC分量最后非零位

    int diff = (int)(data[0] - form);//DC分量差值
    form = data[0];//改变前值
    if (diff == 0)//差值为0
        Out[length++] = DC[0];
    else//差值不为0
    {
        Huff a = getcode(diff);
        Out[length++] = DC[a.length];
        Out[length++] = a;
    }

    while ((last > 0) && (data[last] == 0))//获取最后非零位
        last--;
    for (int i = 1; i <= last; )
    {
        int start = i;
        while ((data[i] == 0) && (i <= last))//获取连续零的个数
            i++;
        int zeronum = i - start;
```

```
        if (zeronum >= 16)//如果连续零个数大于16
        {
            for (int j = 1; j <= zeronum / 16; j++)
                Out[length++] = SIXTEEN_ZEROS;
            zeronum = zeronum % 16;
        }
        Huff a = getcode(data[i]);
        Out[length++] = AC[(zeronum << 4) | a.length];
        Out[length++] = a;
        i++;
    }

    if (last != 63)//最后非零位是否到结束之处
        Out[length++] = EOB;
    Length = length;
}
//编码结果写入文件
void Jpgcreator::write_huff(int Length, int& byte, int& bytepos, const Huff data[],
ofstream& fout)
{
    for (int i = 0; i < Length; i++)
    {
        int value = data[i].value;
        int length = data[i].length - 1;
        while (length >= 0)
        {
            if ((value & highest[length]) != 0)//最高位
                byte = byte | highest[bytepos];//形成一个byte的数据
            length--;
            bytepos--;
            if (bytepos < 0)
            {
                write_one_byte((unsigned char)(byte), fout);//写入文件
                if (byte == 0xFF)
                    write_one_byte((unsigned char)(0x00), fout);//为了避免识别为标识符
                bytepos = 7;//复位
                byte = 0;
            }
        }
    }
}
//总处理中心
void Jpgcreator::jpgcreate()
{
    init();//初始化
    ofstream fout("lena.jpg", ios::binary | ios::out);//打开输出文件
    if (!fout.is_open())
    {
        cout << "illegal";
        exit(-1);
    }
    write_header(fout);//写文件头

    int formY = 0, formCb = 0, formCr = 0;//DC分量前值
```

```
    int byte = 0, bytepos = 7;//要写入的byte
    Huff Out[128];//huffman编码的结果
    int Length;//编码表长度
    double Y[64], Cb[64], Cr[64], tmp[64];
    short ydata[64], cbdata[64], crdata[64];
    for (unsigned int k = 0; k < x; k += 8)
    {
        for (unsigned int l = 0; l < y; l += 8)
        {
            for (int i = 0; i < 8; i++)
            {
                int t = (i + k) * y * 4 + l * 4;
                for (int j = 0; j < 8; j++)
                {
                    double R = data[t++];
                    double G = data[t++];
                    double B = data[t++];
                    t++;
                    Y[i * 8 + j] = (0.299f * R + 0.587f * G + 0.114f * B - 128);
                    Cb[i * 8 + j] = (-0.1687f * R - 0.3313f * G + 0.5f * B);
                    Cr[i * 8 + j] = (0.5f * R - 0.4187f * G - 0.0813f * B);
                }
            }
            //Y的处理
            matrixmultiply(quotient, Y, tmp);//DCT变换
            matrixmultiply(tmp, quotientT, Y);
            compress(Y, ydata, YTable);//量化+Z排
            huffmancoding(formY, Length, ydata, Y_DC_Table, Y_AC_Table, Out);//Huffman编码
            write_huff(Length, byte, bytepos, Out, fout);//编码写入文件
            //Cb的处理（同上）
            matrixmultiply(quotient, Cb, tmp);
            matrixmultiply(tmp, quotientT, Cb);
            compress(Cb, cbdata, CbCrTable);
            huffmancoding(formCb, Length, cbdata, CbCr_DC_Table, CbCr_AC_Table, Out);
            write_huff(Length, byte, bytepos, Out, fout);
            //Cr的处理（同上）
            matrixmultiply(quotient, Cr, tmp);
            matrixmultiply(tmp, quotientT, Cr);
            compress(Cr, crdata, CbCrTable);
            huffmancoding(formCr, Length, crdata, CbCr_DC_Table, CbCr_AC_Table, Out);
            write_huff(Length, byte, bytepos, Out, fout);
        }
    }
    write_two_byte(0xFFD9, fout);//结束
    fout.close();
}
//主函数
int main(int argc, char* argv[])
{
    if (argc != 3)
    {
        cerr << "请确保参数的数目正确" << endl;
        return -1;
    }
```

```
    if (!strcmp(argv[1], "-compress"))//压缩
    {
        Jpgcreator lena(argv[2]);
        lena.jpgcreate();//前往总处理中心
    }

    else if (!strcmp(argv[1], "-read"))//读取
    {
        PicReader imread;
        imread.readPic(argv[2]);
        BYTE* Data = nullptr;
        UINT X, Y;
        imread.getData(Data, X, Y);
        imread.showPic(Data, X, Y);
        delete[] Data;
        Data = nullptr;
    }
    return 0;
}
```

装 订 线