Project5　实验报告

学号：22336289　　姓名：袁鹏湘

实验内容

(1) 图像数据的读取与写入存储，并可通过第三方库调用UI窗口展示图片。
(2) 实现图像数据的压缩存储。首先读取图像，并将图像信息以三元组结构存储。其次，对三元组数据进行压缩存储，以实现数据压缩的功能。最后，将压缩后的数据进行读取，解码，得到原始的图像数据，并进行保存。
(3) 彩色图像转变为灰度图像。将彩色的“color-block.ppm”图像转换为灰度图像，并进行保存和展示。
(4) 实现图像尺寸的缩放。例如将“lena-128-gray.ppm”图像放大为256x256大小；或将“lena-512-gray.ppm”图像缩小为256x256大小。

功能说明

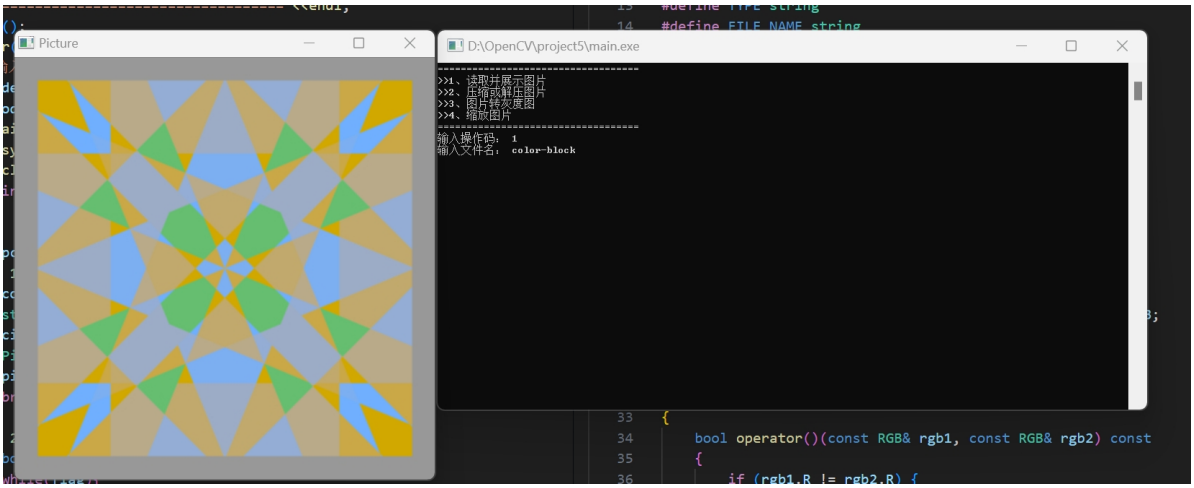
根据界面的指示输入操作代码，选择对应的操作，输入文件名字即可得到结果。（文件默认都是格式规范的ppm文件）

示例

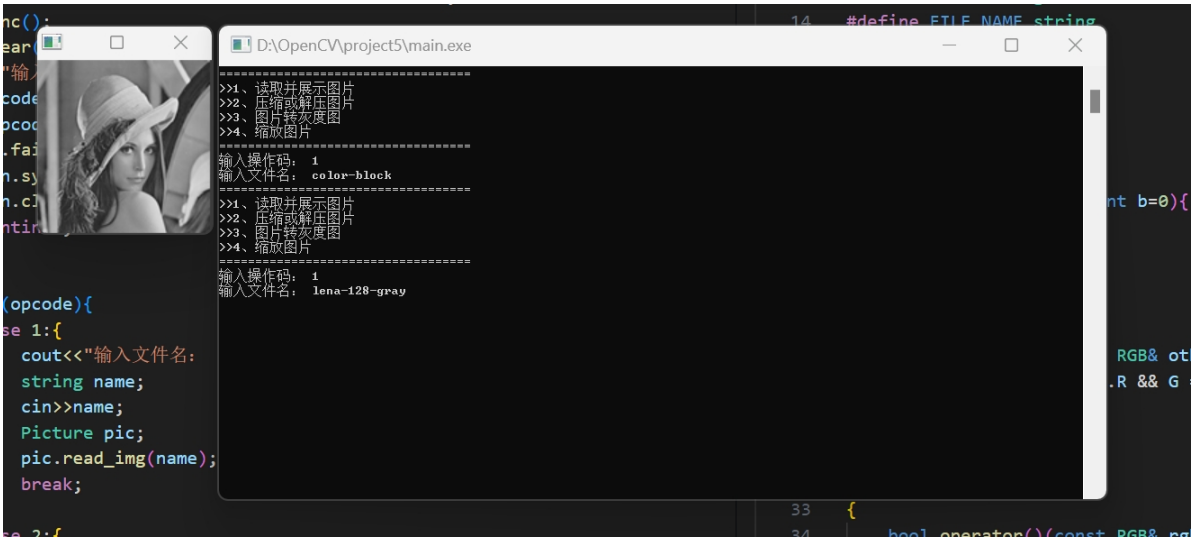
- 示例1

> 选择操作1，以此读取color-block.ppm lena-128-gray.ppm lena-512-gray.ppm

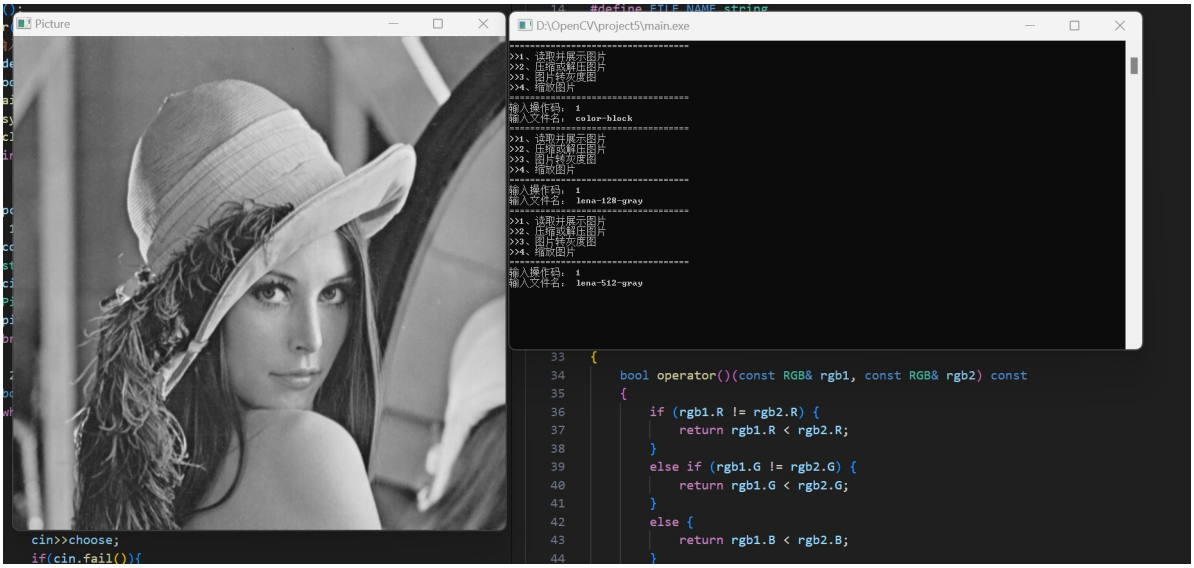
color-block.ppm：



lena-128-gray.ppm：



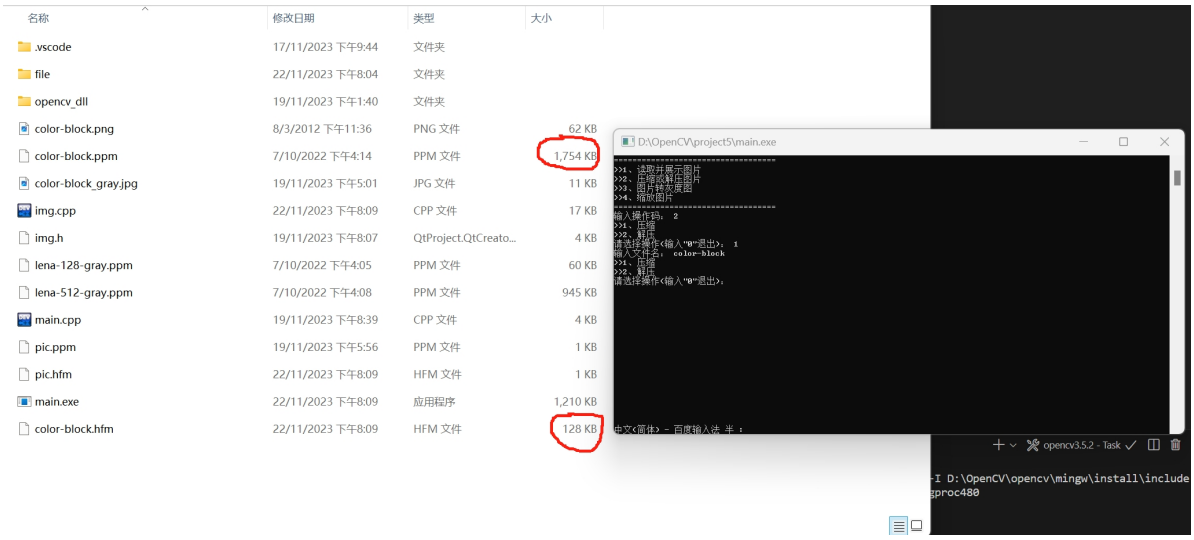
lena-512-gray.ppm：



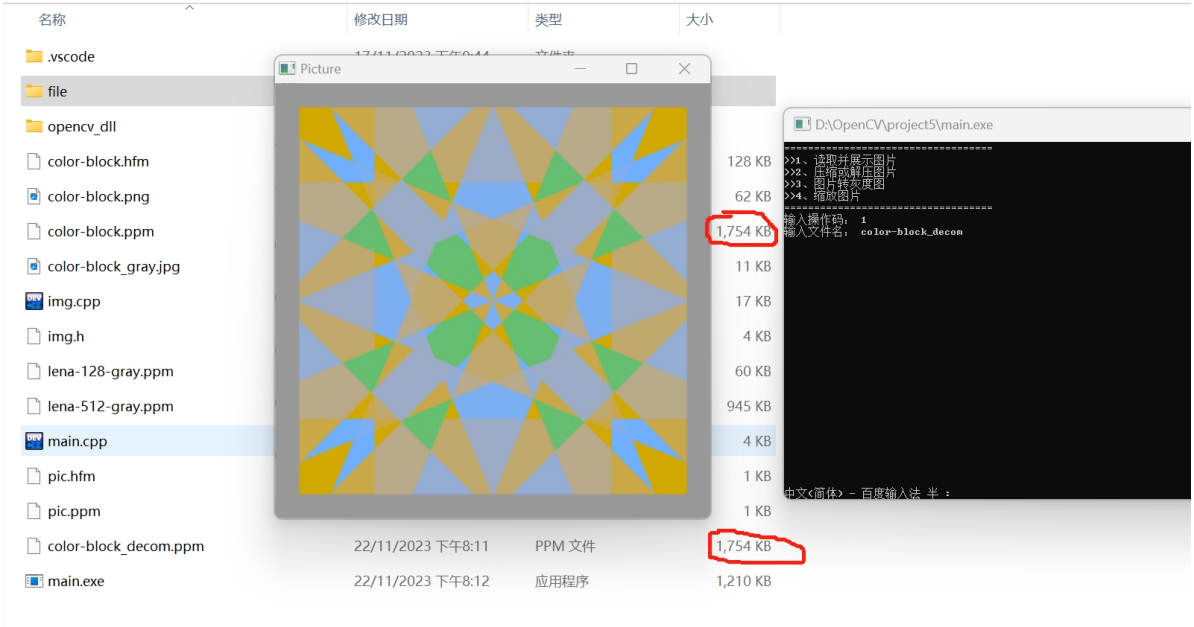
- 示例2

> 选择操作2，并对color-block.ppm进行操作

压缩color-block.ppm：

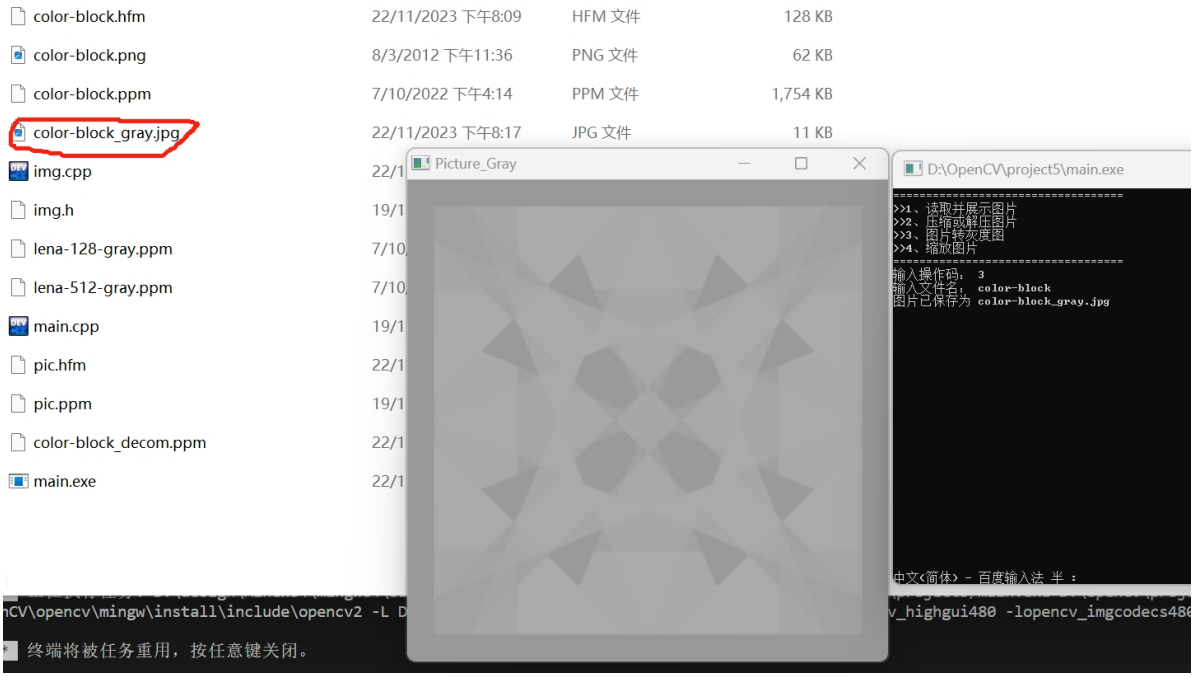


解压color-block.ppm：



- 示例3

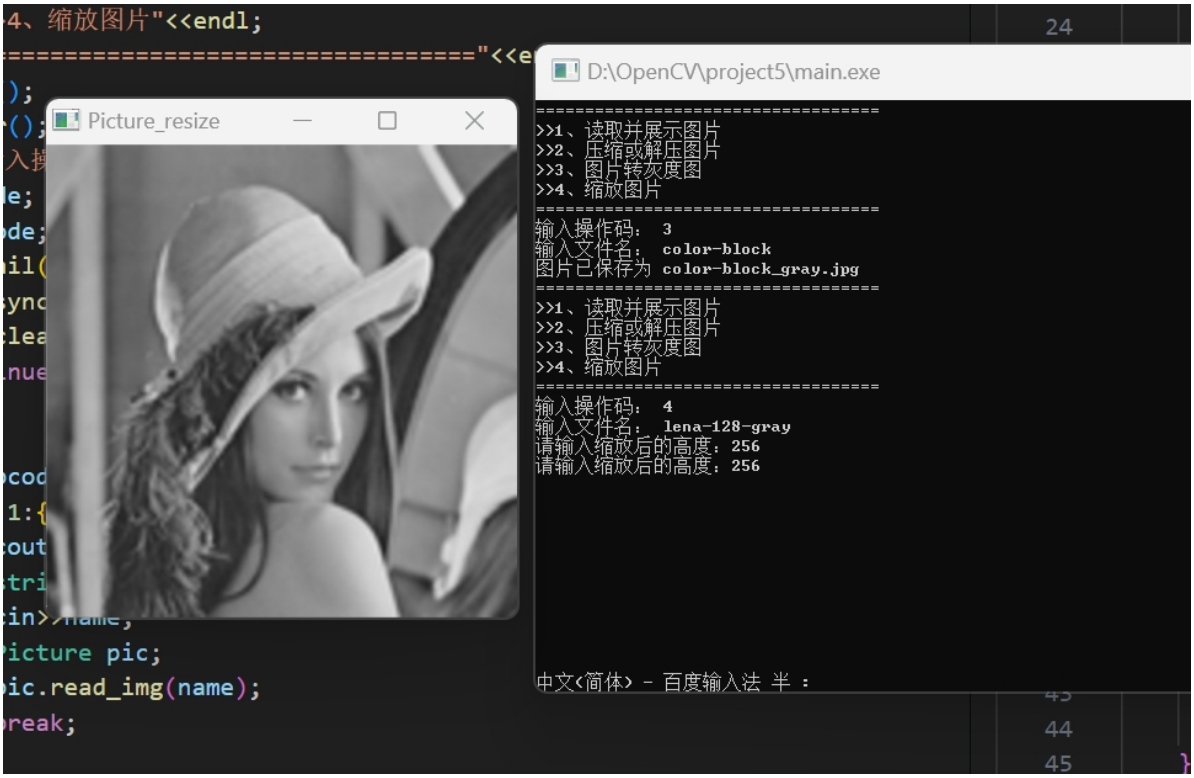
> 选择操作3:，并对color-block.ppm进行操作
> （其他两个ppm文件是P2格式，程序会自动检测并提示无需转换）



- 示例4

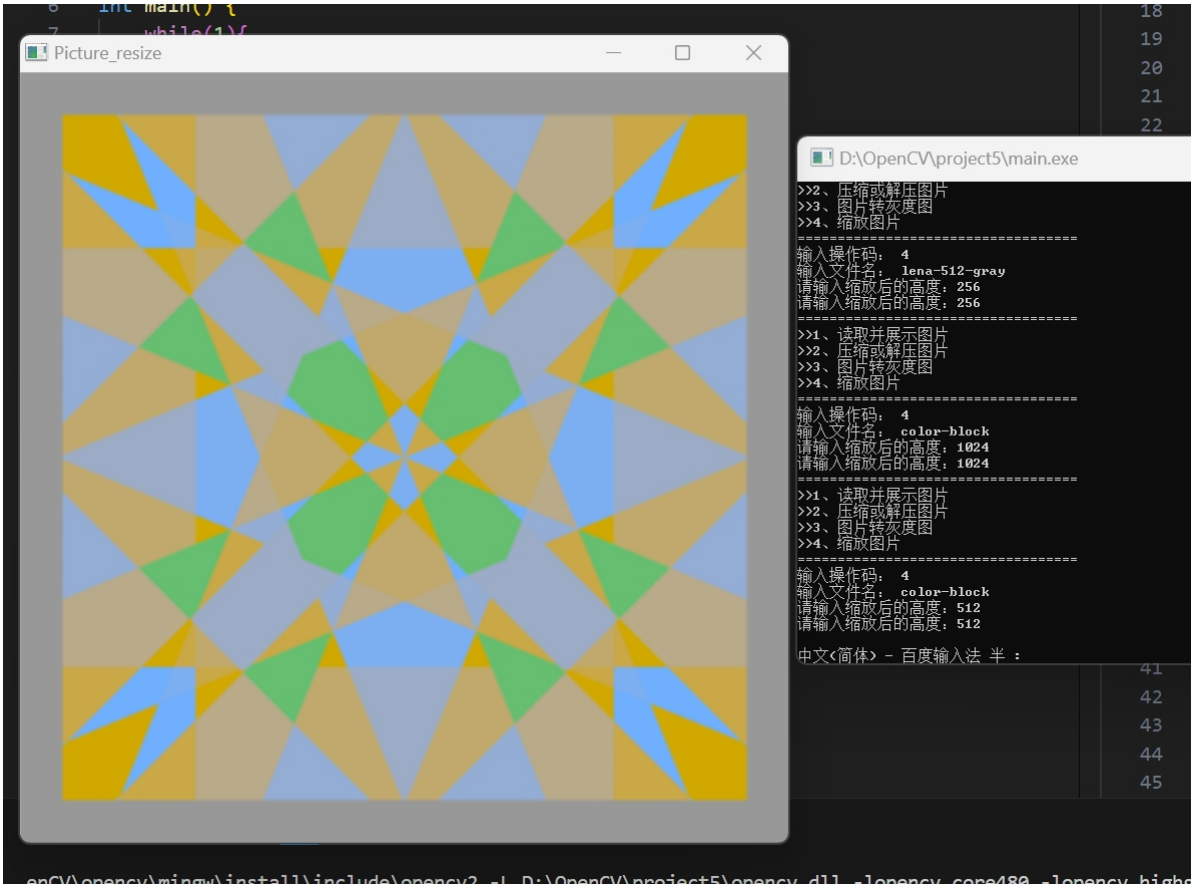
> 选择操作4:，并对 color-block.ppm lena-128-gray.ppm和 lena-512-gray.ppm进行操作

对 lena-128-gray.ppm放大：



对 lena-512-gray.ppm缩小：

对 color-block.ppm放大：



关键代码展示

- **img.h头文件展示**

```
#ifndef _IMG_H_
#define _IMG_H_

#include "opencv2/opencv.hpp"//OpenCV库
#include "vector"
#include "string"
#include "map"
#include "queue"
#include "algorithm"
using namespace std;
using namespace cv;

#define TYPE string
#define FILE_NAME string

struct RGB {
    int R;
    int G;
    int B;

    RGB(int r=0,int g=0,int b=0){
        R=r;
        G=g;
        B=b;
    }//自定义结构体，存储3通道的ppm文件的R,G,B通道的数值

    bool operator==(const RGB& other) const{
```

```
        return R==other.R && G==other.G && B==other.B;
    }//重载运算符，以便于之后用std::map来映射出现频率
};

struct cmp{
    bool operator()(const RGB& rgb1, const RGB& rgb2) const{
        if(rgb1.R!=rgb2.R){
            return rgb1.R<rgb2.R;
        }
        else if(rgb1.G!=rgb2.G){
            return rgb1.G<rgb2.G;
        }
        else{
            return rgb1.B<rgb2.B;
        }
    }
};//自定义比较函数，便于std::map进行排序

template <>
struct std::hash<RGB> {
    std::size_t operator()(const RGB& rgb) const {
        std::size_t h1=std::hash<int>{}(rgb.R);//使用库自带的std::hash<int>来计算
R,G,B的哈希值
        std::size_t h2=std::hash<int>{}(rgb.G);
        std::size_t h3=std::hash<int>{}(rgb.B);
        return h1^(h2<<1)^(h3<<2);//根据自定义的哈希函数来计算RGB的哈希值
    }//自定义哈希函数，接受一个RGB参数并返回哈希值
};//特化哈希函数，便于后面std::unordered_map能够将自定义类型转换为哈希值，实现查找

struct Trible{
    int rol;
    int col;
    RGB val;

    Trible(int r=0,int c=0,RGB v={0,0,0}):rol(r),col(c),val(v){}
};

struct ppm{
    int size;//元素个数
    int total_rol;//总行数
    int total_col;//总列数
    vector<Trible> data;//元素

    ppm(int size=0,int rol=0,int col=0){
        this->size=size;
        data.resize(size);
        total_col=col;
        total_rol=rol;
    }

    void init(int size=0,int rol=0,int col=0){
        this->size=size;
        data.resize(size);
        total_col=col;
        total_rol=rol;
    }
};
```

```
struct TreeNode{
    int pro;//出现频次
    RGB data;//待编码的RGB组

    TreeNode* left;
    TreeNode* right;

    TreeNode(int _pro,RGB _data,TreeNode* _left=nullptr, TreeNode*
_right=nullptr){
        pro=_pro;
        data=_data;
        left=_left;
        right=_right;
    }

};//哈夫曼树节点

struct cmpchar{
    bool operator()(TreeNode* a, TreeNode* b){
        return a->pro > b->pro;
    }
};//便于priority_queue进行排序


/*读取ppm格式的图片的矩阵*/
class Picture{
    public:
        bool check(FILE_NAME name,int x);
        void read_img(FILE_NAME name);//读取图片
        void gray_img(FILE_NAME name);//转灰度图
        void resize_img(FILE_NAME name);//缩放图片

        /*哈夫曼压缩的实现*/
        bool getCount(FILE_NAME name);
        void BuildHuffmanTree();
        void getCodeTable();
        void dfs(TreeNode* root,string code);
        // void encode();
        void compress(FILE_NAME name);
        void decompress(FILE_NAME name);
        void delnode(TreeNode* &node);

    private:
        vector<RGB> rgb;

        ppm pixel_matrix;//矩阵
        TYPE format;//图片类型
        int width;//图片宽度
        int height;//图片高度
        int max_value;//像素最大值
        FILE_NAME pic_name;//文件名

        TreeNode* root;
        unordered_map<RGB,string> CodeTable;//哈夫曼编码表 (rgb)
        map<RGB,int,cmp> FrequencyMap;//出现频次 (rgb)
        int count;//进入统计的RGB个数
};
```

```
#endif  //_IMG_H_
```

- **img.cpp展示**

```
#include "img.h"
#include <iostream>
#include <fstream>

void Picture::delnode(TreeNode* &node){
    if(node==nullptr)
        return;
    delnode(node->left);
    delnode(node->right);
    delete node;
    node=nullptr;
}//删除节点，避免内存泄漏

/*读取ppm文件并展示图片*/
void Picture::read_img(FILE_NAME name){
    ifstream ppm_file(name + ".ppm");

    if (!ppm_file.is_open())
    {
        cout<<"文件打开失败,请检查文件名字是否正确"<<endl;
        return;
    }
    string format;
    ppm_file >> format;//读取格式

    /*P3格式，说明是ASCII编码的3通道图像*/
    if(format=="P3"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;

        cv::Mat img(height,width,CV_8UC3);//构造opencv的Mat对象，行数为高度，列数为宽度，并且数据为8位无符号char类型，有3个通道

        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
                int R,G,B;
                ppm_file>>R>>G>>B;//依次读取R,G,B通道的数值
                img.at<cv::Vec3b>(r,c)=cv::Vec3b(B,G,R);//将RGB向量赋予Mat对象指定的位置
            }
        }

        ppm_file.close();
        imshow("Picture",img);//调用opencv的imshow()函数展示图片
        waitKey(0);
    }
    /*P2格式，说明是ASCII编码的灰度图*/
    else if(format=="P2"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;
```

```
        cv::Mat img(height,width,CV_8UC1);//构造opencv的Mat对象，行数为高度，列数为宽度，并且数据为8位无符号char类型，有1个通道

        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
                int gray;
                ppm_file>>gray;//只用读入灰度值即可
                img.at<uchar>(r,c)=static_cast<uchar>(gray);//将int转为unsigned char类型，并赋予Mat
            }
        }

        ppm_file.close();
        imshow("Picture",img);
        waitKey(0);
    }
    else{
        cout<<"暂不支持的格式"<<endl;
        return;
    }

    // for(int i=0;i<width*height;i++){
    //     ppm_file >> pixel_matrix.data[i].val.R >> pixel_matrix.data[i].val.G >> pixel_matrix.data[i].val.B;
    //     pixel_matrix.data[i].rol = i / width;
    //     pixel_matrix.data[i].col = i % width;
    // }
    // ppm_file.close();

    // cout<<(*CharCount.begin()).first.R;

    // for(int i=0;i<pixel_matrix.size;i++){
    //     cout<<"rol:"<<pixel_matrix.data[i].rol<<" and col:"<<pixel_matrix.data[i].col<<" is:"<<pixel_matrix.data[i].val.R<<" "<<pixel_matrix.data[i].val.G<<" "<<pixel_matrix.data[i].val.B<<endl;
    // }

    // ofstream hfm_file(name+".hfm");
    // hfm_file<<width<<" "<<height<<" "<<max_value<<" ";

    // for(int i=0;i<pixel_matrix.size;i++){
    //     hfm_file<<pixel_matrix.data[i].val.R<<" "<<pixel_matrix.data[i].val.G<<" "<<pixel_matrix.data[i].val.B<<" ";
    // }

}

/*灰度化ppm文件*/
void Picture::gray_img(FILE_NAME name){
    ifstream ppm_file(name + ".ppm");

    if (!ppm_file.is_open())
    {
        cout<<"文件打开失败,请检查文件名字是否正确"<<endl;
        return;
    }
    string format;
    ppm_file >> format;
```

```
    if(format=="P3"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;

        cv::Mat gray_img(height,width,CV_8UC1);//目标文件的Mat

        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
                int R,G,B;
                ppm_file>>R>>G>>B;
                int gray=(R*299+G*587+B*114)/1000;//计算灰度值，采用的是比较常用的公式，为了避免浮点运算，使用int计算更加快速方便，所以统一放大了1000倍后除以1000
                gray_img.at<uchar>(r,c)=static_cast<uchar>(gray);//存入灰度值
            }
        }

        ppm_file.close();
        const string temp=name+="_gray.jpg";
        imwrite(temp,gray_img);//保存文件
        cout<<"图片已保存为 "<<temp<<endl;
        imshow("Picture_Gray",gray_img);//顺便展示灰度图
        waitKey(0);
    }
    else if(format=="P2"){
        cout<<"图片已经是灰度图"<<endl;
        return;
    }
    else{
        cout<<"暂不支持的格式"<<endl;
        return;
    }
}

/*双线性插值缩放图片*/
void Picture::resize_img(FILE_NAME name){
    ifstream ppm_file(name + ".ppm");

    if (!ppm_file.is_open())
    {
        cout<<"文件打开失败,请检查文件名字是否正确"<<endl;
        return;
    }
    string format;
    ppm_file >> format;
    int w,h;
    cout<<"请输入缩放后的高度：";
    cin>>h;
    cout<<"请输入缩放后的高度：";
    cin>>w;

    if(format=="P3"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;

        cv::Mat ori_img(height,width,CV_8UC3);//源文件
        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
```

```
                int R,G,B;
                ppm_file>>R>>G>>B;
                ori_img.at<cv::Vec3b>(r,c)=cv::Vec3b(B,G,R);
            }
        }

        cv::Mat des_img(h,w,CV_8UC3);//目标文件
/*几何平均*/
        for(int r=0;r<h;r++){
            for(int c=0;c<w;c++){
                double ori_y=((double)r+0.5)*(double)height/(double)h-0.5;//计算
缩放后图片像素点所对应原图片的点的y坐标
                double ori_x=((double)c+0.5)*(double)width/(double)w-0.5;//计算缩
放后图片像素点所对应原图片的点的x坐标
/*计算原图点的4个最近点*/
                int left_x=static_cast<int>(ori_x);
                int right_x=left_x+1;
                int down_y=static_cast<int>(ori_y);
                int up_y=down_y+1;

                if((down_y>=height-1)&&(left_x>=width-1)){
                    for(int i=0;i<3;i++){
                        des_img.at<Vec3b>(r,c)[i]=((double)right_x-ori_x)*
((double)up_y-ori_y)*ori_img.at<Vec3b>(down_y,left_x)[i];
                    }
                }//计算出来的原图点区域位于原图之外
                else if(down_y>=height-1){
                    for(int i=0;i<3;i++){
                        des_img.at<Vec3b>(r,c)[i]=((double)right_x-ori_x)*
((double)up_y-ori_y)*ori_img.at<Vec3b>(down_y,left_x)[i]
                        +(ori_x-(double)left_x)*((double)up_y-
ori_y)*ori_img.at<Vec3b>(down_y,right_x)[i];
                    }
                }//计算出来的原图点区域的下y坐标位于原图之外
                else if(left_x>=width-1){
                     for(int i=0;i<3;i++){
                        des_img.at<Vec3b>(r,c)[i]=((double)right_x-ori_x)*
((double)up_y-ori_y)*ori_img.at<Vec3b>(down_y,left_x)[i]
                        +((double)right_x-ori_x)*(ori_y-
(double)down_y)*ori_img.at<Vec3b>(up_y,left_x)[i];
                    }
                }//计算出来的原图点区域的左x坐标位于原图之外
                else{
                    for(int i=0;i<3;i++){
                        des_img.at<Vec3b>(r,c)[i]=((double)right_x-ori_x)*
((double)up_y-ori_y)*ori_img.at<Vec3b>(down_y,left_x)[i]
                        +(ori_x-(double)left_x)*((double)up_y-
ori_y)*ori_img.at<Vec3b>(down_y,right_x)[i]
                        +((double)right_x-ori_x)*(ori_y-
(double)down_y)*ori_img.at<Vec3b>(up_y,left_x)[i]
                        +(ori_x-(double)left_x)*(ori_y-
(double)down_y)*ori_img.at<Vec3b>(up_y,right_x)[i];
                    }
                }//正常情况，使用双线性插值
            }
        }

        ppm_file.close();
```

```
        const string temp=name+="_resize.jpg";
        cout<<"图片已保存为 "<<temp<<endl;
        imwrite(temp,des_img);
        imshow("Picture_resize",des_img);
        waitKey(0);
    }
    else if(format=="P2"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;

        cv::Mat ori_img(height,width,CV_8UC1);
        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
                int gray;
                ppm_file>>gray;
                ori_img.at<uchar>(r,c)=static_cast<uchar>(gray);
            }
        }

        cv::Mat des_img(h,w,CV_8UC1);
        for(int r=0;r<h;r++){
            for(int c=0;c<w;c++){
                double ori_y=((double)r+0.5)*(double)height/(double)h-0.5;
                double ori_x=((double)c+0.5)*(double)width/(double)w-0.5;

                int left_x=static_cast<int>(ori_x);
                int right_x=left_x+1;
                int down_y=static_cast<int>(ori_y);
                int up_y=down_y+1;

                if((down_y>=height-1)&&(left_x>=width-1)){
                    for(int i=0;i<1;i++){
                        des_img.at<uchar>(r,c)=static_cast<uchar>
(((double)right_x-ori_x)*((double)up_y-ori_y)*ori_img.at<uchar>(down_y,left_x));
                    }
                }
                else if(down_y>=height-1){
                    for(int i=0;i<1;i++){
                        des_img.at<uchar>(r,c)=static_cast<uchar>
(((double)right_x-ori_x)*((double)up_y-ori_y)*ori_img.at<uchar>(down_y,left_x)
                        +(ori_x-(double)left_x)*((double)up_y-
ori_y)*ori_img.at<uchar>(down_y,right_x));
                    }
                }
                else if(left_x>=width-1){
                     for(int i=0;i<1;i++){
                        des_img.at<uchar>(r,c)=static_cast<uchar>
(((double)right_x-ori_x)*((double)up_y-ori_y)*ori_img.at<uchar>(down_y,left_x)
                        +((double)right_x-ori_x)*(ori_y-
(double)down_y)*ori_img.at<uchar>(up_y,left_x));
                    }
                }
                else{
                    for(int i=0;i<1;i++){
                        des_img.at<uchar>(r,c)=static_cast<uchar>
(((double)right_x-ori_x)*((double)up_y-ori_y)*ori_img.at<uchar>(down_y,left_x)
                        +(ori_x-(double)left_x)*((double)up_y-
ori_y)*ori_img.at<uchar>(down_y,right_x)
```

```
                        +((double)right_x-ori_x)*(ori_y-
(double)down_y)*ori_img.at<uchar>(up_y,left_x)
                        +(ori_x-(double)left_x)*(ori_y-
(double)down_y)*ori_img.at<uchar>(up_y,right_x));
                    }
                }
            }
        }

        ppm_file.close();
        const string temp=name+="_resize.jpg";
        cout<<"图片已保存为 "<<temp<<endl;
        imwrite(temp,des_img);
        imshow("Picture_resize",des_img);
        waitKey(0);
    }
    else{
        cout<<"暂不支持的格式"<<endl;
        return;
    }
}


/*
以下为哈夫曼压缩的实现：
由于概率论与数理统计已经写过了，这里只是进行部分修改，比如为自定义结构体RGB来设计排序以及哈希函
数；
*/

bool Picture::getCount(FILE_NAME name){
    ifstream ppm_file(name + ".ppm");

    if (!ppm_file.is_open())
    {
        cout<<"文件打开失败,请检查文件名字是否正确"<<endl;
        return false;
    }
    string format;
    ppm_file >> format;

    if(format=="P3"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;
        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
                int R,G,B;
                ppm_file>>R>>G>>B;
                RGB temp(R,G,B);
                rgb.push_back(temp);
                // cout<<"now push struct:<"<<temp.R<<" "<<temp.G<<" "
<<temp.B<<">"<<endl;
            }
        }
        for(auto cur:rgb){
            FrequencyMap[cur]++;//统计RGB出现次数
        }
        count=FrequencyMap.size();//计数
        // for(auto it:FrequencyMap){
```

```
        // std::cout << "Frequency of color: " <<(it.first).R<<" "
<<(it.first).G<<" "<<(it.first).B<<" -> "<<it.second<< std::endl;
        // }
    }
    else if(format=="P2"){
        int width,height,maxVal;
        ppm_file>>width>>height>>maxVal;

        for (int r=0;r<height;r+=1) {
            for (int c=0;c<width;c+=1) {
                int gray;
                ppm_file>>gray;
                RGB temp(gray,0,0);//方便起见，即便1通道灰度图也可以假装为3通道，只要剩下
两个通道填0，压缩时不对其压缩即可
                rgb.push_back(temp);
            }
        }
        for(auto cur:rgb){
            FrequencyMap[cur]++;//统计RGB出现次数
        }
        count=FrequencyMap.size();//计数
    }
    else{
        cout<<"暂不支持的格式"<<endl;
        return false;
    }
    return true;
}

void Picture::BuildHuffmanTree(){
    /*为所有RGB元素构建节点，并且全部入队*/
    priority_queue<TreeNode*, vector<TreeNode*>, cmpchar> node_queue;
    for(auto it=FrequencyMap.begin();it!=FrequencyMap.end();it++){
            node_queue.push(new TreeNode((*it).second,
(*it).first,nullptr,nullptr));
            // cout<<(*it).first.R<<" "<<(*it).first.G<<" "<<(*it).first.B<<":"
<<(*it).second<<endl;
    }
    /*每次取出2个最小的频度的节点，合并后并且插回队列中*/
    while(node_queue.size()>1){
        TreeNode* temp1=node_queue.top();
        node_queue.pop();
        // cout<<"temp1:"<<temp1->data.R<<":"<<temp1->pro<<endl;
        TreeNode* temp2=node_queue.top();
        node_queue.pop();
        // cout<<"temp2:"<<temp2->data.R<<":"<<temp2->pro<<endl;
        TreeNode* newnode = new TreeNode(temp1->pro + temp2-
>pro,RGB(0,0,0),temp1,temp2);
        node_queue.push(newnode);
    }

    root=node_queue.top();
}

/*获取哈夫曼编码表*/
void Picture::getCodeTable(){
    dfs(root,"");
```

```
    // for(auto it:CodeTable){
    //     cout<<CodeTable.size()<<endl;
    //     cout<<it.first.R<<" "<<it.first.G<<" "<<it.first.B<<"对应的编码："
<<it.second<<endl;
    // }
}

void Picture::dfs(TreeNode* root,string code){
    if(!root)
        return;
    if((!root->left)&&(!root->right)){
        CodeTable[root->data]=code;//当走到叶子结点时，保存改节点RGB元素对应的哈夫曼编码
        // cout<<"数据"<<root->data.R<<" "<<root->data.G<<" "<<root->data.B<<"的编
码："<<code<<endl;
        return;
    }
    dfs(root->left,code+"0");//往左走为0
    dfs(root->right,code+"1");//往右走为1
}

/*压缩数据*/
void Picture::compress(FILE_NAME name){
    ifstream inFile(name+".ppm");
    if (!inFile.is_open())
    {
        cout<<"文件打开失败,请检查文件名字是否正确"<<endl;
        return;
    }
    ofstream outFile(name+".hfm",ios::binary);
    if(!outFile.is_open()){
        cout<<"错误，文件已经存在或者权限不足无法创建"<<endl;
        return;
    }


    inFile.seekg(0,ios::beg);
    string code="";
    unsigned char bitdata=0;
    int addition=0;

    string type;
    int width,height,max_color;
    inFile>>type>>width>>height>>max_color;
    // cout<<type<<width<<height<<max_color;
    if(type=="P3"){
        int r,g,b;
        while(!inFile.eof())
        {
            inFile>>r>>g>>b;
            RGB temp(r,g,b);
            code+=CodeTable[temp];//不断读取数据，并且根据map映射的哈夫曼编码来往后延长总
的编码code
        }
        // cout<<"对文本压缩后的编码："<<code<<endl;

        addition=code.length()%8 ? (8-code.length()%8) : 0;//附加的0的个数
        //  cout<<"应该附加的0的个数："<<addition<<endl;
```

```
        for(int i=0;i<addition;i++){
            code+="0";
        }//附加0，方便后面按字节处理编码

        // cout<<"对文本压缩后的编码："<<code<<endl;

        /*写入附加0个数*/
        outFile.write(reinterpret_cast<char*>(&addition), sizeof(int));

        //  cout<<"字符的个数："<<count<<endl;

        /*写入RGB元素个数*/
        outFile.write(reinterpret_cast<char*>(&count), sizeof(int));

        /*写入相关文件信息*/
        outFile.write(type.c_str(), type.length());//类型
        outFile.write(reinterpret_cast<char*>(&width),sizeof(int));//宽度
        outFile.write(reinterpret_cast<char*>(&height),sizeof(int));//高度
        outFile.write(reinterpret_cast<char*>(&max_color),sizeof(int));//最大值

        /*将RGB以及对应频率写入文件,方便解压时构造哈夫曼树*/
        for(auto it=FrequencyMap.rbegin();it!=FrequencyMap.rend();it++){
            int a,b,c;
            RGB temp=(*it).first;
            a=temp.R;
            b=temp.G;
            c=temp.B;
            outFile.write(reinterpret_cast<const char*>(&a),sizeof(int));
            outFile.write(reinterpret_cast<const char*>(&b),sizeof(int));
            outFile.write(reinterpret_cast<const char*>(&c),sizeof(int));
            outFile.write(reinterpret_cast<char*>(&(*it).second),sizeof(int));
        }

        /*正式压缩编码code为2进制形式*/
        int len=code.length();
        char c=0;//初始化char为0
        for(int i=0;i<len;i++){
            c <<= 1;//每读一位，左移1位
            if(code[i]=='1')
                c |= 1;//如果编码为1，要在最低位+1
            if((i+1)%8==0){
                outFile.write(reinterpret_cast<char*>(&c),sizeof(char));//满8位即
char满了，就执行char写入文件里
            }
        }
    }//由于补齐了0，一定是所有char都能写满的
    else if(type=="P2"){
        int gray;
        while(!inFile.eof())
        {
            inFile>>gray;
            RGB temp(gray,0,0);
            code+=CodeTable[temp];
        }

        // cout<<"对文本压缩后的编码："<<code<<endl;

        addition=code.length()%8 ? (8-code.length()%8) : 0;//附加的0的个数
```

```
    //  cout<<"应该附加的0的个数："<<addition<<endl;

        for(int i=0;i<addition;i++){
            code+="0";
        }
        // cout<<"对文本压缩后的编码："<<code<<endl;

        /*写入附加0个数*/
        outFile.write(reinterpret_cast<char*>(&addition), sizeof(int));

    //  cout<<"字符的个数："<<count<<endl;
        /*写入RGB元素个数*/
        outFile.write(reinterpret_cast<char*>(&count), sizeof(int));

        outFile.write(type.c_str(), type.length());//类型
        outFile.write(reinterpret_cast<char*>(&width),sizeof(int));//宽度
        outFile.write(reinterpret_cast<char*>(&height),sizeof(int));//高度
        outFile.write(reinterpret_cast<char*>(&max_color),sizeof(int));//最大值

        /*将RGB以及对应频率写入文件*/
        for(auto it=FrequencyMap.rbegin();it!=FrequencyMap.rend();it++){
            int a,b,c;
            RGB temp=(*it).first;
            a=temp.R;
            b=temp.G;
            c=temp.B;
            outFile.write(reinterpret_cast<const char*>(&a),sizeof(int));//只写
R，即gray，其他是0没必要写入；
            // outFile.write(reinterpret_cast<const char*>(&b),sizeof(int));
            // outFile.write(reinterpret_cast<const char*>(&c),sizeof(int));
            outFile.write(reinterpret_cast<char*>(&(*it).second),sizeof(int));
        }

        int len=code.length();
        char c=0;
        for(int i=0;i<len;i++){
            c <<= 1;
            if(code[i]=='1')
                c |= 1;
            if((i+1)%8==0){
                outFile.write(reinterpret_cast<char*>(&c),sizeof(char));
            }
        }
    }
    else{
        cout<<"暂不支持的类型"<<endl;
        return;
    }

}

/*解压文件*/
void Picture::decompress(FILE_NAME name){
    ifstream inFile(name+".hfm",ios::binary);
    if (!inFile.is_open())
    {
        cout<<"文件打开失败,请检查文件名字是否正确"<<endl;
        return;
```

```
    }
    ofstream outFile(name+"_decom.ppm");
    if(!outFile.is_open()){
        cout<<"错误，文件已经存在或者权限不足无法创建"<<endl;
        return;
    }

    int addition;
    int cnt;
    inFile.read(reinterpret_cast<char*>(&addition),sizeof(int));
//  cout<<"读取的addition："<<addition<<endl;

    inFile.read(reinterpret_cast<char*>(&cnt),sizeof(int));
//  cout<<"读取的count："<<cnt<<endl;


    int width,height,max_color;
    char type[3];
    inFile.read(type, 2);//类型
    inFile.read(reinterpret_cast<char*>(&width),sizeof(int));//宽度
    inFile.read(reinterpret_cast<char*>(&height),sizeof(int));//高度
    inFile.read(reinterpret_cast<char*>(&max_color),sizeof(int));//最大值

    /*头部信息写入*/
    outFile<<type<<endl<<width<<" "<<height<<endl<<max_color<<endl;
    string format=type;
    if(format=="P3"){
        RGB data;
        int pro;
        FrequencyMap.clear();
        /*读取头部信息保存的频率表*/
        for(int i=0;i<cnt;i++){
            int a,b,c;
            inFile.read(reinterpret_cast<char*>(&a),sizeof(int));
            inFile.read(reinterpret_cast<char*>(&b),sizeof(int));
            inFile.read(reinterpret_cast<char*>(&c),sizeof(int));
            RGB temp(a,b,c);
            inFile.read(reinterpret_cast<char*>(&pro),sizeof(int));
            // cout<<"读取到的ch:"<<a<<" "<<b<<" "<<c<<",对应的频数："<<pro<<endl;
            FrequencyMap[temp]=pro;//还原频率表
        }

        BuildHuffmanTree();//建树

        // for(auto it:CodeTable){
        //     cout<<it.first.R<<" "<<it.first.G<<" "<<it.first.B<<"对应的编码："
<<it.second<<endl;
        // }

        char c;
        string code="";
        while(inFile.get(c)){
            for(int i=0;i<8;i++){
                if((c&128)==128)
                    code+="1";
                else
                    code+="0";
                c<<=1;
```

```
            }
        }//重新读取编码

        // cout<<"解压后的编码："<<code<<endl;

        int len=code.length();
        TreeNode* cur=root;
        for(int i=0;i<=len-addition;){
            if(cur->left||cur->right){
                if(code[i]=='0')
                    cur=cur->left;
                else
                    cur=cur->right;
                i++;
            }
            else{
                outFile<<cur->data.R<<" "<<cur->data.G<<" "<<cur->data.B<<" ";
                cur=root;
            }
        }
    }//往文件写入解码后RGB对应的R,G,B
    else if(format=="P2"){
        RGB data;
        int pro;
        FrequencyMap.clear();
        for(int i=0;i<cnt;i++){
            int a;
            inFile.read(reinterpret_cast<char*>(&a),sizeof(int));//只用读gray即可
            // inFile.read(reinterpret_cast<char*>(&b),sizeof(int));
            // inFile.read(reinterpret_cast<char*>(&c),sizeof(int));
            RGB temp(a,-1,-1);
            inFile.read(reinterpret_cast<char*>(&pro),sizeof(int));
            // cout<<"读取到的ch:"<<a<<" "<<b<<" "<<c<<",对应的频数："<<pro<<endl;
            FrequencyMap[temp]=pro;
        }

        BuildHuffmanTree();

        // for(auto it:CodeTable){
        //     cout<<it.first.R<<" "<<it.first.G<<" "<<it.first.B<<"对应的编码："
<<it.second<<endl;
        // }

        char c;
        string code="";
        while(inFile.get(c)){
            for(int i=0;i<8;i++){
                if((c&128)==128)
                    code+="1";
                else
                    code+="0";
                c<<=1;
            }
        }

        // cout<<"解压后的编码："<<code<<endl;

        int len=code.length();
```

```
        TreeNode* cur=root;
        for(int i=0;i<=len-addition;){
            if(cur->left||cur->right){
                if(code[i]=='0')
                    cur=cur->left;
                else
                    cur=cur->right;
                i++;
            }
            else{
                outFile<<cur->data.R<<" ";
                cur=root;
            }
        }
    }


}

/*检测文件格式是否正确，不是很重要的代码*/
bool Picture::check(FILE_NAME name,int x){
    switch(x){
        case 0:{
            ifstream in(name+".ppm");
            if(!in.is_open())
                return false;
            else
                return true;
            in.close();
            break;
        }
        case 1:{
            ifstream in(name+".hfm");
            if(!in.is_open())
                return false;
            else
                return true;
            in.close();
            break;
        }
        default:{
            return false;
        }
    }
    return false;
}
```