```
#include "stdafx.h"
#include "funtions.h"
#include <stdio.h>

using namespace std;


int readImage()
{

std::cout << "-----------start to read Image-------------" << std::endl;

//读入单张图片，路径可替换成自己的图片的路径
cv::Mat srcMat = imread("../testImages\\butterfly.jpg");

//读取图片的一些信息
// Mat是否为空，可以判断读图是否成功
std::cout << "empty:" << (srcMat.empty() ? "the Mat is empty,fail to read" : "not empty") << std::endl;
if (srcMat.empty())return -1;

//在Mat中
//cols 是 列数 相当于 width
//rows 是 行数 相当于 height
//行数
std::cout << "rows:" << srcMat.rows << std::endl;
//列数
std::cout << "cols:" << srcMat.cols << std::endl;
//维度，普通图片为2维
std::cout << "dims:" << srcMat.dims << std::endl;

// Size是OpenCV内部定义的数据类型
std::cout << "size[]:" << srcMat.size().width << "," << srcMat.size().height << std::endl;

// 深度id
// 用来度量每一个像素中【每一个通道】的精度，depth数值越大，精度越高。在
//Opencv中，Mat.depth()得到的是一个0~6的数字，分别代表不同的位数，对应关系如下：
//opencv中，由于使用Mat.at访问数据时，必须正确填写相应的数据类型，
//Mat的类型定义方法
//The definition is as follows:
//CV_（位） + （数据类型） + （通道数量）
//如, CV_32FC1 表示 32位 float型单通道,
//OpenCV中的数据类型与C的数据类型的对应关系。
/*
uchar   CV_8U 0
char    CV_8S 1
ushort  CV_16U 2
short   CV_16S 3
int     CV_32S 4
float   CV_32F 5
double  CV_64F 6
*/
std::cout << "depth (ID):" << srcMat.depth() << std::endl;

// channel数，如灰度图为单通道，RGB图为3通道
std::cout << "channels:" << srcMat.channels() << std::endl;

// Mat中一个元素的size(byte数),矩阵一个元素占用的字节数，
//数据类型是
//CV_8UC1， elemSize==1，1 byte；
//CV_8UC3/CV_8SC3， elemSize==3；3 byte
//CV_16UC3/CV_16SC3， elemSize==6；6 byte
//即elemSize==字节数x通道数；矩阵一个元素占用的字节数，
std::cout << "elemSize:" << srcMat.elemSize() << "[byte]" << std::endl;
// Mat中一个元素的一个通道的size(byte数)，矩阵元素一个通道占用的字节数，
//eleSize1==elemSize/channels；
std::cout << "elemSize1 (elemSize/channels):" << srcMat.elemSize1() << "[byte]" << std::endl;

//元素的总数，如果是图像，即为像素个数
std::cout << "total:" << srcMat.total() << std::endl;
// step （byte数）
//Mat矩阵中每一行的“步长”，以字节为基本单位，每一行中所有元素的字节总量
//cols*elemSize=cols*eleSize1*channels
std::cout << "step:" << srcMat.step << "[byte]" << std::endl;
// 一个step的channel总数，每行的channel数
std::cout << "step1 (step/elemSize1):" << srcMat.step1() << std::endl;
// 该Mat在内存上是否连续
std::cout << "isContinuous:" << (srcMat.isContinuous() ? "true" : "false") << std::endl;
// 是否为子矩阵
std::cout << "isSubmatrix:" << (srcMat.isSubmatrix() ? "true" : "false") << std::endl;
```

```
//读入单张图片，加参数0，表示读入，并转换成灰度图
cv::Mat gryMat = imread("../testImages\\butterfly.jpg", 0);
if (srcMat.empty())return -1;
//保存图片
imwrite("../testImages\\gray-butterfly.jpg",gryMat);

//显示图片
imshow("src", srcMat);
imshow("gray", gryMat);

//显示图片，必须要加waitKey()，否则无法显示图像
//waitKey(0),无限地显示窗口，直到任何按键按下
//如果是其他数字，如waitKey(25)表示25毫秒，然后关闭。
waitKey(0);

//关闭所有窗口
destroyAllWindows();

return 0;
}

//通过OpenCV读取视频
int readVideo()
{
std::cout << "-----------start to read Video-------------" << std::endl;

//读取本地视频，OpenCV可以读取本地视频文件，摄像头，及连续的图像文件
//VideCapture为opencv定义的视频数据的类，实际是底层对ffmpeg的封装实现的

//-----------------------读取视频文件--------------------------
//实例化VideoCapture类，名为cap，并打开（）中的视频
//也可以通过 capVideo.open("../testImages\\vtest.avi"); 打开
//如果 capVideo.open(0)则打开默认摄像头，参数0为摄像头的id
VideoCapture capVideo("../testImages\\vtest.avi");

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}

//读取视频的一些属性，更多参数可参考videoio_c.h中得定义
cout << "parameters" << endl;
cout << "width： " << capVideo.get(CV_CAP_PROP_FRAME_WIDTH) << endl;
cout << "heigth： " << capVideo.get(CV_CAP_PROP_FRAME_HEIGHT) << endl;
cout << "frames： " << capVideo.get(CV_CAP_PROP_FRAME_COUNT) << endl;
cout << "fps： " << capVideo.get(CV_CAP_PROP_FPS) << endl;

//保存文件初始化，VideoWriter为OpenCV中定义的视频保存类
VideoWriter writer;
//选择编码方式
int codec = CV_FOURCC('M', 'J', 'P', 'G');
// 输出的视频地址及名字
string filename = "../testImages\\saved.avi";
//定义帧率
double fps = capVideo.get(CV_CAP_PROP_FPS);
//保存的视频的尺寸,此处尺寸缩小一半
cv::Size vSize;
vSize.width = capVideo.get(CV_CAP_PROP_FRAME_WIDTH) / 2;
vSize.height = capVideo.get(CV_CAP_PROP_FRAME_HEIGHT) / 2;

//打开视频流
writer.open(filename, codec, fps, vSize);

Mat frame;
Mat resizeFrame;
Mat grayFrame;

while (1) {
//视频流中读取图像
capVideo >> frame;

if (frame.empty()) {
cout << "Unable to read frame!" << endl;
destroyAllWindows();
return -1;
}

//保存到视频流，由于视频文件尺寸降为1/2，frame尺寸也要减半
resize(frame,resizeFrame,vSize);
writer.write(resizeFrame);
//可以接各种处理
cvtColor(frame, grayFrame, CV_RGB2GRAY);

imshow("frame",frame);
imshow("resizeFrame",resizeFrame);
imshow("gray",grayFrame);

//显示图片，延时30ms，必须要加waitKey()，否则无法显示图像
//等待键盘相应，按下ESC键退出
if (waitKey(30) == 27){
destroyAllWindows();
break;
}
}

destroyAllWindows();
return 0;
}

//读取连续图片
int readSequence()
{
//(eg. `img_%02d.jpg`, which will read samples like `img_00.jpg, img_01.jpg, img_02.jpg, ...`)
VideoCapture capSequence("../testImages\\sequence\\left%02d.jpg");

if (!capSequence.isOpened())
{
cerr << "Unable to open the image sequence!\n" << endl;
return 1;
}

cv::Mat frame;
while (1) {
//视频流中读取图像
capSequence >> frame;

if (frame.empty()) {
cout << "Unable to read frame!" << endl;
destroyAllWindows();
return -1;
}

imshow("frame", frame);

waitKey(200);

}

return 0;

}



//Mat类的创建方法，及初始化示例
int createMat()
{
//---创建Mat---
//cols 是 列数 相当于 width
//rows 是 行数 相当于 height
int cols = 4;
int rows = 3;
int type = CV_32S;
int dataArray[] = { 0,  1,  2,  3,
10, 11, 12, 13,
10, 11, 12, 13 };

cv::Mat mat1_0; //实例化，此操作并不在内存上开辟空间
cv::Mat mat2; //实例化，此操作并不在内存上开辟空间
cv::Mat mat3; //实例化，此操作并不在内存上开辟空间


//几种方法，进行初始化定义尺寸和类型，并开辟空间
mat1_0.create(rows, cols, type);
mat2.create(Size(cols, rows), type);
mat3.create(mat1_0.size(), mat1_0.type());

//通过指针对mat1初始化
cv::Mat mat1_1(rows, cols, CV_32S, &dataArray);
```

```
//如果mat1的保存空间连续，则拷贝数组的数据给mat1
//Mat的数据实际保存在成员数组 data 里面
if (mat1_0.isContinuous()) {
memcpy(mat1_0.data, dataArray, sizeof(int)*cols*rows);
}

//生成随机数
//均一分布的随机数，[0,256)
cv::randu(mat2, cv::Scalar(0), cv::Scalar(256));
// 正太分布的随机数，mean=128, stddev=10
cv::randn(mat3, cv::Scalar(128), cv::Scalar(10));

std::cout << "m1_0:" << std::endl << mat1_0 << std::endl << std::endl;
std::cout << "m1_1:" << std::endl << mat1_1 << std::endl << std::endl;
std::cout << "m2:" << std::endl << mat2 << std::endl << std::endl;
std::cout << "m3:" << std::endl << mat3 << std::endl << std::endl;

//---创建Mat---
// 创建数据类型为64F, channels=10, 3x3 的2维矩阵
cv::Mat mat4(3, 3, CV_64FC(10));
//也可以通过CV_MAKETYPE()获得赋值的参数，本例中CV_MAKETYPE(CV_64F, 10)==78
cv::Mat mat5(3, 3, CV_MAKETYPE(CV_64F, 10));

//创建channels=2，int型，2x2矩阵，并赋值，其他数据类型可查matx.hpp中的定义
cv::Mat mat6 = (cv::Mat_<cv::Vec2i>(2, 2) << cv::Vec2i(1, 1), cv::Vec2i(2, 4),
 cv::Vec2i(3, 9), cv::Vec2i(4, 16));

std::cout << "m6:" << std::endl << mat6 << std::endl << std::endl;

// 5×4矩阵， 5行×4列，元素均为1
cv::Mat mat7 = cv::Mat::ones(5, 4, CV_8U);
// 5×4矩阵， 5行×4列，元素均为3
cv::Mat mat8 = cv::Mat::ones(5, 4, CV_8U) * 3;
// 5×4矩阵， 5行×4列，元素均为0
cv::Mat mat9 = cv::Mat::zeros(5,4, CV_8U);
// 3×3矩阵， 3行×3列，单位矩阵
cv::Mat mat10 = cv::Mat::eye(3, 3, CV_8U);

std::cout << "m7:" << std::endl << mat7 << std::endl << std::endl;
std::cout << "m8:" << std::endl << mat8 << std::endl << std::endl;
std::cout << "m9:" << std::endl << mat9 << std::endl << std::endl;
std::cout << "m10:" << std::endl << mat10 << std::endl << std::endl;


return 0;

}

//Mat的复制方法
//Mat的复制，有深复制及浅复制的分别
int copyMat()
{
//生成一个3×3的Mat
cv::Mat m1 = (cv::Mat_<double>(3, 3) << 1, 2, 3, 4, 5, 6, 7, 8, 9);

//浅复制，实质只是把m1的内存地址赋值给m_shallow
//两个Mat在内存中是同一块数据
cv::Mat m_shallow = m1;

//深复制，clone和copyTo，为m_deep1及m_deep2在内存中开辟空间，并且复制内容
cv::Mat m_deep1 = m1.clone();
cv::Mat m_deep2;
m1.copyTo(m_deep2);

std::cout << "m1=" << m1 << std::endl << std::endl;
std::cout << "m_shallow=" << m_shallow << std::endl << std::endl;
std::cout << "m_deep1=" << m_deep1 << std::endl << std::endl;
std::cout << "m_deep2=" << m_deep2 << std::endl << std::endl;

// 修改m1的(0,0)位置的数值，注意观察修改以后，其他几个Mat的内容
m1.at<double>(0, 0) = 100;

std::cout << "m1=" << m1 << std::endl << std::endl;
std::cout << "m_shallow=" << m_shallow << std::endl << std::endl;
std::cout << "m_deep1=" << m_deep1 << std::endl << std::endl;
std::cout << "m_deep2=" << m_deep2 << std::endl << std::endl;


//定义ROI并复制
//ROI(region of interest)感兴趣区域，即需要被处理的区域
//Rect是opencv中定义的矩形数据类型
```

```
//读入单张图片，路径可替换成自己的图片的路径
cv::Mat srcMat = imread("../testImages\\butterfly.jpg");
cv::Mat roiMat;
cv::Rect roi;
roi.x = 0;
roi.y = 0;
roi.width = srcMat.cols / 2;
roi.height = srcMat.rows / 2;

//定义mask并复制
//mask即遮罩，用来屏蔽掉图像中的部分区域
//mask的格式为uchar格式的mat，黑色部分表示需要屏蔽的，白色表示不需要遮蔽
//生成mask
cv::Mat mask= cv::Mat::zeros(srcMat.size(), CV_8U);
rectangle(mask,roi,Scalar(255),-1);

cv::Mat maskedMat;

//复制ROI区域
srcMat(roi).copyTo(roiMat);

//带mask复制
srcMat.copyTo(maskedMat, mask);

imshow("src",srcMat);
imshow("mask",mask);
imshow("masked image",maskedMat);
imshow("roi",roiMat);

waitKey(0);

return 0;
}


//利用Mat进行一些基本运算
int calcMat()
{
//创建Mat
cv::Mat m1 = (cv::Mat_<double>(3, 3) << 1, 2, 3, 4, 5, 6, 7, 8, 9);
std::cout << "m1=" << m1 << std::endl << std::endl;

//基本四则运算
cv::Mat m2 = m1 + 3;
cv::Mat m3 = m1 * 3;
cv::Mat m4 = m1 / 3;

std::cout << "m2=" << m2 << std::endl << std::endl;
std::cout << "m3=" << m3 << std::endl << std::endl;
std::cout << "m4=" << m4 << std::endl << std::endl;

//mat和mat的运算
cv::Mat m5 = m1 + m1;
//m6和m2相同位置的数值相乘
cv::Mat m6 = m1.mul(m2);
//相乘后，再乘以系数
cv::Mat m7 = m1.mul(m2, 2);

std::cout << "m5=" << m5 << std::endl << std::endl;
std::cout << "m6=" << m6 << std::endl << std::endl;
std::cout << "m7=" << m7 << std::endl << std::endl;


//要确保运算Mat的类型和尺寸相同，如果不同，则抛出异常
//Mat类型不同
cv::Mat m8 = (cv::Mat_<int>(3, 3) << 1, 2, 3, 4, 5, 6, 7, 8, 9);
try {
std::cout << m1 / m8 << std::endl;
}
catch (cv::Exception e) {
std::cout << std::endl;
}

//Mat的尺寸不同
cv::Mat m9 = (cv::Mat_<double>(2, 2) << 1, 2, 3, 4);
try {
std::cout << m9 / m1 << std::endl;
}
catch (cv::Exception e) {
// ...
std::cout << std::endl;
}


return 0;
}

//一些基本的线性代数操作
int calcLinearAlg()
{
//向量的内积和外积
cv::Vec3d v1(1, 2, 3);
cv::Vec3d v2(3, 4, 5);

//内积
double v_dot = v1.dot(v2);
//外积
cv::Vec3d v_cross = v1.cross(v2);
std::cout << "v_dot=" << v_dot << std::endl;
cv::Mat tmp(v_cross);
std::cout << "v_cross=" << tmp << std::endl;

//求范数
// 6x1
cv::Mat m1 = (cv::Mat_<double>(6, 1) << 1, 5, 3, -1, -3, -5);
// 向量(3,4)
cv::Point p1(3, 4);
// 6维度向量的 L-2范数
double norm_m1 = cv::norm(m1);
// 2维度向量的 L-2范数
double norm_p1 = cv::norm(p1);

std::cout << std::endl;
std::cout << "norm(m1)=" << norm_m1 << std::endl;
std::cout << "norm(p1)=" << norm_p1 << std::endl << std::endl;

// 通过2维坐标，计算极坐标，即大小和角度
std::cout << "calc Polar" << std::endl;
//创建4组2维坐标
cv::Mat x = (cv::Mat_<double>(4, 1) << 0, 1, 4, 1);
cv::Mat y = (cv::Mat_<double>(4, 1) << 1, 1, 3, 1.7320504);
cv::Mat magnitude, angle;
cv::cartToPolar(x, y, magnitude, angle, true);

for (int i = 0; i<4; ++i) {
std::cout << "(" << x.at<double>(i) << ", " << y.at<double>(i) << ") ";
std::cout << "mag=" << magnitude.at<double>(i) << ", angle=" << angle.at<double>(i) << "[deg]" << std::endl;
}

std::cout << std::endl;
// 通过大小和角度，计算2维坐标
std::cout << "calc Cartesian" << std::endl;
cv::Mat mag2 = (cv::Mat_<double>(4, 1) << 1, 1.41421, 5, 2);
cv::Mat ang2 = (cv::Mat_<double>(4, 1) << 90, 45, 36.8699, 60);

cv::Mat x2, y2;
cv::polarToCart(mag2, ang2, x2, y2, true); // in degrees

for (int i = 0; i<4; ++i) {
std::cout << "(" << x2.at<double>(i) << ", " << y2.at<double>(i) << ") ";
std::cout << "mag=" << mag2.at<double>(i) << ", angle=" << ang2.at<double>(i) << "[deg]" << std::endl;
}

std::cout << std::endl;

return 0;
}

//求解线性方程
int solveLinearEquations()
{
//独立方程数和未知数相同时
//  x +  y +  z = 6
// 3x + 2y - 2z = 1
// 2x - y  + 3z = 9
//左边
cv::Mat lhand = (cv::Mat_<double>(3, 3) << 1, 1, 1, 3, 2, -2, 2, -1, 3);
//右边
cv::Mat rhand = (cv::Mat_<double>(3, 1) << 6, 1, 9);

//高斯消去法求解
cv::Mat ans;
```

```
cv::solve(lhand, rhand, ans);

std::cout << "Gaussian elimination" << std::endl;
std::cout << "(x,y,z) = " << ans << std::endl << std::endl;
//独立方程数 多于 未知数数量时
//通过最小二乘法求解
//  x +  y = 3
// 3x + 4y = 8
// -x - 2y = 2
std::cout << "the least square method" << std::endl;
cv::Mat lhand2 = (cv::Mat_<double>(3, 2) << 1, 1, 3, 4, -1, -2);
cv::Mat rhand2 = (cv::Mat_<double>(3, 1) << 3, 8, 2);

cv::Mat x;
//通过SVD求解最小二乘法
//方程组左侧，方程组右侧，输出，求解方法
cv::solve(lhand2, rhand2, x, cv::DECOMP_SVD);

std::cout << "(x,y) = " << x << std::endl;
std::cout << "norm(lhand2*x-rhand2)=" << norm(lhand2*x - rhand2) << std::endl << std::endl;

return 0;
}
```