使用说明书

目录

PPM、PGM 和 PBM 图像处理.....

YUV 图像处理.....

RAW 图像处理.....

BMP 图像处理.....

其它处理.....

高级算子.....

PPM、PGM 和 PBM 图像处理

void OTSUBinarization(char* input,char* output)	OTSU 二值化。input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。
void PPMtoBMP(char* input,char* output,int bpp)	PPM 图像转为 BMP 图像。bpp 是 BMP 图像的色深。
void BMPtoPPM(char* input,char* output)	BMP 图像转 PPM 图像。
void PPMtoBMP1(char* input,char* output,int bpp)	PPM 图像转为 BMP 图像。bpp 是 BMP 图像的色深。
void BMPtoPPM1(char* input,char* output)	BMP 图像转 PPM 图像。
void BMPtoPGM(char* input, char* output)	BMP 转 PGM。
void BMPtoPPM2(char* input, char* output)	BMP 转 PPM。
void PPMtoPGM(char* input, char* output)	PPM 转 PGM。
void BlurPPM(char* input, char* output)	PPM 图像滤波。
void BlurPGM(char* input, char* output)	PGM 图像滤波。
void SegmentsOTSUBinarization(char* input,char* output)	OTSU 二值化划分。input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。
void P3PPMBlur(char* input,char* output)	PPM 图像模糊，input 是输入文件名，output 是输出文件名。支持 P3 格式的 PPM 图像。
unsigned char** ReadPBM(char* input)	读取 PBM 图像并返回图像数据。input 是要读取的 PBM 图像文件名。支持 P4

	格式的 PBM 图像。
void WritePBM(unsigned char** Input,char* output)	保存 PBM 图像。Input 是输入的图像数据，output 是输出文件名。支持 P4 格式的 PBM 图像。
void PGMHistogramEqualization(char* input,char* output)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。

PPMImage* ReadPPM(char* input)	PPM 图像读取，input 是要读取的 PPM 图像文件名。支持 P6 格式的 PPM 图像。
	需要引入的结构体：
	typedef struct {
	unsigned char red, green, blue;
	//像素的颜色由 RGB（红/绿/蓝）表示
	} PPMPixel;
	typedef struct {
	unsigned int width, height;
	// 图像的宽度和高度（以像素为单位）
	PPMPixel *data;
	// 构成图像的像素
	} PPMImage;

void WritePPM(char* output,PPMImage* img)	PPM 图像保存， output 是输出的 PPM 图像文件名，img 是输入的图像数据。支持 P6 格式的 PPM 图像。
	需要引入的结构体：
	typedef struct {
	unsigned char red, green, blue;
	//像素的颜色由 RGB（红/绿/蓝）表示
	} PPMPixel;
	typedef struct {
	unsigned int width, height;
	// 图像的宽度和高度（以像素为单位）
	PPMPixel *data;
	// 构成图像的像素
	} PPMImage;

void InvertColor(char* input,char* output)	负滤波器,input 是输入文件名,output 是输出文件名。支持 P6 格式的 PPM 图像。
void GrayFilter(char* input,char* output)	灰度过滤器，input 是输入文件名，output 是输出文件名。支持 P6 格式的 PPM 图像。

void SepiaFilter(char* input,char* output)	乌贼墨过滤器，input 是输入文件名，output 是输出文件名。支持 P6 格式的 PPM 图像。
void AdjustSaturation(char* input,char* output,double a)	调整图像饱和度,input 是输入文件名，output 是输出文件名。a 是目标饱和度，如 a=30。支持 P6 格式的 PPM 图像。
void Resize(char* input,char* output,unsigned int NewWidth, unsigned int NewHeight)	调整图像大小，input 是输入文件名，output 是输出文件名。NewWidth 和 NewHeight 分别是输出图像的宽和高。支持 P6 格式的 PPM 图像。
void AdjustHue(char* input,char* output,int a)	调整图像的色调,input 是输入文件名，output 是输出文件名。a 是目标色调，如 a=125。支持 P6 格式的 PPM 图像。
void AdjustBrightness(char* input,char* output,double a)	调整图像亮度，input 是输入文件名，output 是输出文件名。a 是目标亮度，如 a=60。支持 P6 格式的 PPM 图像。
void AdjustContrast(char* input,char* output,double a)	调整图像对比度,input 是输入文件名，output 是输出文件名。a 是目标对比度，如 a=60。支持 P6 格式的 PPM 图像。
void AdjustBlur(char* input,char* output,double a)	通过 sigma 因子模糊图像，input 是输入文件名，output 是输出文件名。a 是 sigma 因子,如 a=5。支持 P6 格式的 PPM 图像。
void MeanGrayFilter(char* input,char* output,double a)	平均灰度滤波器,input 是输入文件名，output 是输出文件名。a 是平均值系数，如 a=3。支持 P6 格式的 PPM 图像。
void Pixelate(char* input,char* output,unsigned int a)	像素化，input 是输入文件名，output 是输出文件名。a 是幅度值，如 a=8。支持 P6 格式的 PPM 图像。
void Rotate(char* input,char* output,short a)	旋转图像,input 是输入文件名,output 是输出文件名。a 是旋转的角度，如 a=45。支持 P6 格式的 PPM 图像。
void GammaCorrection(char* input,char* output,double a)	伽马校正,input 是输入文件名,output 是输出文件名。a 是 gamma 数,如 a=0.5。支持 P6 格式的 PPM 图像。
void GrayAndChannelSeparation(char* input,char* Grayoutput,char* Routput,char* Goutput,char* Boutput)	生成灰度图以及 RGB 通道分离，input 是输入的 P6 格式的 PPM 图像；Grayoutput 是输出的灰度图文件名，Routput、Goutput 和 Boutput 分别是输出的 R、G 和 B 通道的图像文件名，输出都是 PGM 格式文件。
void PGMBin(char* input,char* output,int threshold)	灰度图像二值化，输入是灰度图像，输入和输出都是 PGM 文件，threshold 是阈值，如 threshold=125。

void Brightening(char* input,char* output,int a)	彩色图像增亮，输入和输出都是 P6 格式的 PPM 图像，a 是增亮系数，如 a=80。
void GrayBrightening(char* input,char* output,int a)	灰度图像增亮，输入和输出都是 PGM 图像，a 是增亮系数，如 a=80。
void PPMFilter(char* input,char* output)	彩色图像滤波，输入和输出都是 P6 格式的 PPM 文件。
void PGMGrayFilter(char* input,char* output)	灰度图像滤波，输入和输出都是 PGM 图像。
void PPMtoBMP(char* input,char* output)	PPM 图像转 BMP 图像，input 是输入文件名，output 是输出文件名。支持 P6 格式的 PPM 图像。
void PGMOtsuThreshold(string input,char* output)	大津阈值法，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。
void PGMRotated(char* input,char* output,int width,int height,int channels,double theta)	channels 是输入图像的通道，theta 是旋转弧度，参考：theta=45.0*3.1415926/180。
void XCorner(char* input,char* output,int width,int height,int channels,double theta)	channels 是输入图像的通道。
void YCorner(char* input,char* output,int width,int height,int channels,double theta)	channels 是输入图像的通道。
void Smooth(char* input,char* output,int width,int height,int channels,float sigma_x,float sigma_y,double theta)	channels 是输入图像的通道，sigma_x 是 X 方向的模糊系数，sigma_y 是 Y 方向的模糊系数。
void PGMLocalisedOtsuThreshold(string input,char* output)	局部大津阈值，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。
void PGMSauvolaThreshold(string input,char* output,double a,double b,double c)	索沃拉阈值，支持 P5 格式的 PGM 图像。a、b 和 c 的参考值如：a=0.01,b=15,c=225。
void PGMThreshold(string input,char* output,int thresh)	阈值法，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。thresh 是阈值，如：thresh=5。
float Repair1(char* input,char* output,float var,float threshold,int nbLevels,float a)	图像修复，var 是噪声方差，threshold 是阈值，nbLevels 是要处理的级别数，a=10。返回 ISNR。
float Repair2(char* input,char* output,float var,float threshold,int nbLevels,float a)	图像修复，var 是噪声方差，threshold 是阈值，nbLevels 是要处理的级别数，a=10。返回 ISNR。
void LowPassFilterRepair1(char*	低通滤波图像修复，a=10，b=6，

input,char* output,int size_filter,float var,int nb_iterations,int nbLevels,float a,int b)	nbLevels=3，size_filter 是低通过滤器的大小，var 是噪声方差，nb_iterations 是 Landweber 的迭代数。
void LowPassFilterRepair2(char* input,char* output,int size_filter,float var,int nb_iterations,int nbLevels,float a,int b)	低通滤波图像修复，a=10，b=6，nbLevels=3，size_filter 是低通过滤器的大小，var 是噪声方差，nb_iterations 是 Landweber 的迭代数。
float LowPassFilterRepair3(char* input,char* output,int size_filter,float var,int nb_iterations,int nbLevels,int pas,float a,int b)	低通滤波图像修复，a=10，b=6，nbLevels=3，pas=1，size_filter 是低通过滤器的大小，var 是噪声方差，nb_iterations 是 Landweber 的迭代数。返回 ISNR。
void Repair1(char* input,char* output,int M,float a)	图像修复，a=0.0，M 是分解的层次数，如 M=3。
void Repair2(char* input,char* output,int M,float a)	图像修复，a=0.0，M 是分解的层次数，如 M=3。
void MakeNoise1(char* input,char* output,int size_filter)	制造噪声，size_filter 是低通滤波器的宽度。
void MakeNoise2(char* input,char* output,int nb_iterations,int pas)	制造噪声,nb_iterations 是 Landweber 的迭代数，pas=1。
void MakeNoise3(char* output,int height,int width,float var)	制造噪声，height 是输出图像的高，width 是输出图像的宽，var 是噪声方差。
void MakeNoise4(char* input,char* output,int nb_iterations,int pas)	制造噪声,nb_iterations 是 Landweber 的迭代数，pas=1。
void ImageReconstruction(char* input,char* output,int maxDepth,int threshold,int tx,int ty)	图像重建，支持 PGM 文件。参考：maxDepth=80，threshold=50，tx=0，ty=0。

YUV 图像处理

void YUVsuperposition(char* input1,char* input2,char* output,int width,int height,unsigned char Y_BLACK,unsigned char U_BLACK,unsigned char V_BLACK)	YUV420 叠加，Y_BLACK、U_BLACK 和 V_BLACK 用于将原图中的黑色变成透明，参考：Y_BLACK=16，U_BLACK=128，V_BLACK=128。
void YUVsuperposition(char* input1,char* input2,char* output,int width,int	YUV444 叠加，Y_BLACK、U_BLACK 和 V_BLACK 用于将

height,unsigned char Y_BLACK,unsigned char U_BLACK,unsigned char V_BLACK)	原图中的黑色变成透明，参考：Y_BLACK=16，U_BLACK=128，V_BLACK=128。
void YUVsuperposition(char* input1,char* input2,char* output,int width,int height,unsigned char Y_BLACK,unsigned char U_BLACK,unsigned char V_BLACK)	yuv444p 直接叠加到 yuv420p 上，不做转换，Y_BLACK、U_BLACK 和 V_BLACK 用于将原图中的黑色变成透明，参考：Y_BLACK=16，U_BLACK=128，V_BLACK=128。
void YUV444toYUV420(char* input,char* output,int height,int width)	YUV444 转 YUV420，height 是输入的 YUV444 文件的高，width 是输入的 YUV444 文件的宽。
void YUV444toYUV420(char* input,char* output,int height,int width,int frames)	YUV444 转 YUV420，height 和 width 是输入文件的高和宽，frames 是要输入文件中操作的帧序号。
void YUVsuperposition(char* input1,char* input2,char* output,int width,int height,unsigned char Y_BLACK,unsigned char U_BLACK,unsigned char V_BLACK)	YUV444 转到 YUV420 上的叠加，Y_BLACK、U_BLACK 和 V_BLACK 用于将原图中的黑色变成透明，参考：Y_BLACK=16，U_BLACK=128，V_BLACK=128。
void YUVEdgeProcessingY(char* input,char* output,int width,int height,double k)	YUV 边缘处理，input 是输入文件名，output 是输出文件名。width 和 height 是输入图像的宽和高。参考：k=0.5。
void YUVEdgeProcessingU(char* input,char* output,int width,int height,double k)	YUV 边缘处理，input 是输入文件名，output 是输出文件名。width 和 height 是输入图像的宽和高。参考：k=0.5。
void YUVEdgeProcessingV(char* input,char* output,int width,int height,double k)	YUV 边缘处理，input 是输入文件名，output 是输出文件名。width 和 height 是输入图像的宽和高。参考：k=0.5。
void BMPLoadedIntoYUV(char* inputBMP,char* inputYUV,char* output,int YUVwidth,int YUVheight,int depth,bool mt)	YUV 加载 BMP，inputBMP 是输入的 BMP 图像，inputYUV 是输入的 YUV 图像，inputYUV 起到容器的作用，

	YUVwidth 和 YUVheight 是输入的 YUV 图像的宽和高，参考：depth=12，mt=true。
void YUVEdgeProcessingHorizontalDirection(char* input,char* output,int width,int height,double k)	YUV 仅水平方向的边缘处理，input 是输入文件名，output 是输出文件名。width 和 height 是输入图像的宽和高。参考：k=0.7。
void YUVVieoEdgeProcessing(char* input,char* output,int width,int height,int frame,int max_frame)	YUV 视频文件边缘处理，input 是输入文件名，output 是输出文件名。width 和 height 是输入图像的宽和高，frame 是要处理的帧序号，max_frame 是最大帧序号。
void YUVScale(char* input,char* output,int inputWidth,int inputHeight,int outputWidth,int outputHeight)	缩放 yuv420 图像，参考：inputWidth=1280，inputHeight=720，outputWidth=128，outputHeight=72。
void NoiseTreatment(char* input,char* output,int width,int height,int TWICEwidth,int TWICEheight)	YUV 噪声处理。
void NoiseTreatment(char* input,char* output,int width,int height,int frame,int max_frame)	YUV 噪声处理。

RAW 图像处理

unsigned char** RAWRead(char* input,int height,int width)	RAW 图像读取，返回像素数据。
void RAWWrite(unsigned char** input,char* output,int height,int width)	接收像素数据保存为 RAW 图像。
void MBVQ(char* input,char* output,int width,int height)	MBVQ 效果，input 是输入文件名，output 是输出文件名。width 和 height 是输出图像的宽和高。
void RAWtoPPM_red(char* input,char* output,int width, int height,DebayerAlgorithm algo)	RAW 转为 PPM 后提取红色通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR 或 LINEAR。支持 RAW12 格式。 需引入以下枚举： enum DebayerAlgorithm { NEARESTNEIGHBOUR, LINEAR

	};

void RAWtoPPM_green1(char* input,char* output,int width, int height,DebayerAlgorithm algo)	RAW 转为 PPM 后提取绿色 1 通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR 或 LINEAR。
	支持 RAW12 格式。
	需引入以下枚举：
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RAWtoPPM_green2(char* input,char* output,int width, int height,DebayerAlgorithm algo)	RAW 转为 PPM 后提取绿色 2 通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR 或 LINEAR。
	支持 RAW12 格式。
	需引入以下枚举：
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RAWtoPPM_blue(char* input,char* output,int width, int height,DebayerAlgorithm algo)	RAW 转为 PPM 后提取蓝色通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR 或 LINEAR。
	支持 RAW12 格式。
	需引入以下枚举：
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RAWtoPPM(char* input,char* output,int width, int height,DebayerAlgorithm algo)	RAW 转为 PPM，参考：width=4096，height=3072 ，algo=NEARESTNEIGHBOUR 或 LINEAR。
	支持 RAW12 格式。
	需引入以下枚举：
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RawPowerTransformation(char* input,char* output,int width,int height,int c,float v)	幂次变换，input 是输入的 RAW 图像文件名，output 是输出的 RAW 图像文件名，width 是输入图像的

	宽，height 是输入图像的高。默认 c=1，v=0.6。支持 RAW 图像。
void RAWAvgFilter(char* input,char* output,int ROWS,int COLS,int M,float mask[3][3])	平均滤波器，input 是输入文件名，output 是输出文件名。ROWS 是图像的行大小，COLS 是图像的列大小，M 是滤波相关参数，如 M=1；mask 是滤波器模板。支持 RAW 图像。 参考模板： float mask[3][3] = {{0.1111,0.1111,0.1111}, {0.1111,0.1111,0.1111}, {0.1111,0.1111,0.1111}};
void RawImageInversion(char* input,char* output,int width,int height)	图像反相,input 是输入的 RAW 图像文件名，output 是输出的 RAW 图像文件名,width 是输入图像的宽，height 是输入图像的高。支持 RAW 图像。
void RawHistogramEqualization(char* input,char* output,int width,int height)	直方图均衡化，input 是输入的 RAW 图像文件名，output 是输出的 RAW 图像文件名，width 是输入图像的宽，height 是输入图像的高。支持 RAW 图像。
void RAWHistogramEqualization(char* input,char* output,int width,int height)	RAW 直方图均衡化，width 和 height 是输入图像的宽和高。
void RAWMedianFilter(char* input,char* output,int ROWS,int COLS,int M,int sequence[9])	中值滤波，input 是输入文件名，output 是输出文件名。ROWS 是图像的行，COLS 是图像的列，M 是滤波相关参数，如 M=1。支持 RAW 图像。 参考模板： int sequence[9]={0,0,0,0,0,0,0,0,0};
void RawtoBmp1(char* input, char* output,unsigned long Width, unsigned long Height)	RAW 图像转为 BMP 图像，input 是输入文件名，output 是输出文件名。Width 和 Height 是输入文件的宽和高。
void RawToBmp(char* input,char* output,int imageWidth,int imageHigth)	RAW 图像转为 BMP 图像，input 是输入文件名，output 是输出文件名。支持宽和高相等的图像。

void RGBtoCMY(string input,string output1,string output2,int height,int width,int NumberChannels,int a)	RGB 转 CMY,output1 是输出的 CMY 模型图像名，output2 是输出的 CMY 模型图像的单个通道的图像名；height 和 width 是输入图像 input 的高和宽;NumberChannels 是图像的通道数，如通道数为 3；a=0 表示生成 cyan 模型图像,a=1 表示生成 magenta 模型图像，a=2 表示生成 yellow 模型图像。支持 RAW 文件。
void RGBtoHSI(char* input,char* output)	RGB 模型转为 HIS 模型，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void SobelOperation1(char* input,char* output,int width,int height)	Sobel 算子。
void SobelOperation2(char* input,char* output,int width,int height)	Sobel 算子。
void CyanGray(char* input,char* output,int width,int height)	青色灰度图像。
void MagentaGray(char* input,char* output,int width,int height)	品红灰度图像。
void YellowGray(char* input,char* output,int width,int height)	黄色灰度图像。
void GrayLightness(string input,string output,int height,int width,int NumberChannels)	彩色转灰度。
void GrayAverage(string input,string output,int height,int width,int NumberChannels)	彩色转灰度。
void GrayLuminosity(string input,string output,int height,int width,int NumberChannels)	彩色转灰度。
void Transfer(char* input,char* output,int width,int height)	传递函数。
void Homography(char* input1,char* input2,char* input3,char* output,int width,int height,int newwidth,int newheight)	单应。
void MovieEffect(char* input,char* output,int width,int height)	电影效果。
void Dither(string input,string output,int height,int width,int	抖动，参考：method=1，bayerMatrixNumber=2 ，

NumberChannels,int method,int bayerMatrixNumber,int numberOfTones)	numberOfTones=2。
void AssimilateChannels(string input,string output,int height,int width,int NumberChannels,int method,int bayerMatrixNumber,int numberOfTones)	抖动，参考：method=1，bayerMatrixNumber=2，numberOfTones=2。
void FixedThresholdMethod(char* input,char* output,int width,int height)	抖色处理，固定阈值法。
void RandomThresholdMethod(char* input,char* output,int width,int height)	抖色处理，随机阈值法。
void DitherMatrixMethod(char* input,char* output,int width,int height,int N)	抖色处理,抖动矩阵法,默认N=2。
void NormalizedLogBuffer1(char* input,char* output,int width,int height)	对数变换，规范化对数。
void NormalizedLogBuffer2(char* input,char* output,int width,int height)	对数变换，规范化对数。
void TernaryGrayLevel1(char* input,char* output,int width,int height)	三值灰度。
void TernaryGrayLevel2(char* input,char* output,int width,int height)	三值灰度。
void BestEdgeMap1(char* input,char* output,int width,int height)	最佳边贴图。
void BestEdgeMap2(char* input,char* output,int width,int height)	最佳边贴图。
void Skeletonize(char* input,char* output,int width,int height)	骨架化。
void GrayLuminosity(string input,string output,int height,int width,int NumberChannels)	转换为灰度。
void DifferentiateImageSobelFilter(string input,string output,int height,int width,int NumberChannels)	微分图像以获得边缘。
void DifferentiateImageSobelFilterAndRGBtoCMY(string input,string output,int	求反图像以获得边缘映射。

height,int width,int NumberChannels)	
void EdgeDetectionSobelFilter(string input,string output,int height,int width,int NumberChannels,int threshold)	使用 Sobel 过滤器获取图像的边缘贴图，参考：threshold=50。
void RemoveSpeckles(string input,string output,int height,int width,int NumberChannels,int threshold,int background,int numberOfIterations,type_morphing morphingOperation)	参考：threshold=-100，background=255，numberOfIterations=0，morphingOperation=THINNING。 本函数需添加以下结构体： typedef enum type_morphing { SHRINKING = 0x1, THINNING = 0x2, SKELETONIZING = 0x3 } type_morphing;
void DoubleDifferentiatingGetEdges(string input,string output,int height,int width,int NumberChannels)	双微分图像获取边缘。
void GetTernaryMap(string input,string output,int height,int width,int NumberChannels,int threshold,string output_Histogram,bool writeHistogramToFile)	创建三元边贴图。参考：threshold=50，writeHistogramToFile=false。
void SeparableDiffusion(char* input,char* output,int width,int height)	可分离扩散。
void Denoising(char* input1,char* input2,char* output,int width,int height)	去除噪声。
void Luminosity(char* input,char* output,int width,int height)	亮度调整。
void Average(char* input,char* output,int width,int height)	平均化。
void MinMax(char* input,char* output,int width,int height)	最小与最大。
void Shrink(char* input,char* output,int width,int height)	收缩。
void BilinearTransformation(char* input,char* output,int width,int height,int newwidth,int newheight)	双线性变换。
void BilinearInterpolation(string input,string output,int height,int	双线性插值，参考：targetHeight=200，

width,int NumberChannels,int targetHeight,int targetWidth)	targetWidth=200。
void DitherMatrixMethod(char* input,char* output,int width,int height,int N)	四级抖动，默认 N=2。
int ObjectsInImages(string input,string output,int height,int width,int NumberChannels,int threshold,float starSize,int a)	图像中的对象，参考：threshold=128，starSize=1.5833，a=1 表示确保前背景为黑色。支持 RAW 文件。
void Dewarped1(char* input,char* output,int width,int height,int Offset,double a,double b)	脱蜡。a 是在输出图像中检查半径是否<=a，然后再进行扭曲，参考：Offset=256，a=256.5，b=0.5。
void Dewarped2(char* input,char* output,int width,int height,int Offset,double a,double b,double coeffx[12],double coeffy[12])	脱蜡。a 是在输出图像中检查半径是否<=a，然后再进行扭曲，参考：Offset=256，a=256.5，b=0.5。 脱蜡规范： double coeffx[12] = { 1.00056776e+00, -5.68880703e-04, -1.13998357e-03, 1.00056888e+00,-5.65549579e-04,-1.13554790e-03, 9.99434446e-01 , 5.66658513e-04 , 1.13110351e-03 , 9.99433341e-01 , 5.67767429e-04 , 1.13553921e-03 }; double coeffy[12] = {-5.67763072e-04, 1.00056888e+00, 1.13998357e-03, 5.68880703e-04, 9.99434450e-01, -1.13554790e-03, 5.65553919e-04, 9.99433341e-01, -1.13110351e-03, -5.66658513e-04, 1.00056777e+00, 1.13553921e-03};

void TextureSegmentation1(char* input,char* output,int width,int height,int K,int N)	纹理分割，默认 K=6，N=100。
void TextureSegmentation2(char* input,char* output,int width,int height,int K,int N)	纹理分割，默认 K=6，N=100。
void TextureClassification(vector<string> filename,char* output,int width,int height,int K,int N,int a)	纹理分类,a 是要分类的图像的数量，如 filename 里有 3 个图像名称，则 a=3；output 是分类结果文件，格式为 txt 的文本文件；默认 K=4，N=1000。
void ErrorDiffusion1(char* input,char* output,int width,int height)	误差扩散。
void ErrorDiffusion2(char* input,char* output,int width,int height)	误差扩散。
void ErrorDiffusion3(char* input,char* output,int width,int height)	误差扩散。
void ErrorDiffusion(string input,string output,int height,int width,int NumberChannels,int method,int kernelSize,int numberOfTones,bool useFilter)	误差扩散，参考：method=1，kernelSize=3，numberOfTones=2，useFilter=false。支持 RAW 文件。
void Thin(char* input,char* output,int width,int height)	图像细化。
void BinaryMorphologicalFilteringComplete(string input,string output,int height,int width,int NumberChannels,int threshold,int numberOfIterations,type_morphing morphingOperation)	形态滤波。 参考：threshold=100，numberOfIterations=0，morphingOperation=(type_morphing)3。支持 RAW 文件。
void OilPainting(char* input,char* output,int width,int height,int N)	油画效果，默认 N=2。
void OilPainting1(char* input,char* output,int width,int height,int N)	油画效果，默认 N=2。
void OilPaintingEffect1(string input,string output,int height,int width,int NumberChannels,int colorBits,bool bitsOK,int kernelSize,bool kernelSizeOK)	油画效果，参考：colorBits=3，bitsOK=true，kernelSize=9，kernelSizeOK=true。支持 RAW 文件。
void OilPaintingEffect2(string	油画效果，参考：colorBits=3，

input,string output,int height,int width,int NumberChannels,int colorBits,bool bitsOK,int kernelSize,bool kernelSizeOK)	bitsOK=true， kernelSize=9，kernelSizeOK=true。支持 RAW 文件。
void OilPaintingEffect3(string input,string output,int height,int width,int NumberChannels,int colorBits,bool bitsOK,int kernelSize,bool kernelSizeOK)	油画效果，参考：colorBits=3，bitsOK=true， kernelSize=9，kernelSizeOK=true。支持 RAW 文件。
void AverageFiltering(char* inputfile,char* outputfile,int width,int height)	3*3 平均值滤波。
void GeometricMeanFiltering(char* inputfile,char* outputfile,int width,int height)	3*3 几何均值滤波。
void MedianFiltering(char* inputfile,char* outputfile,int width,int height)	中值滤波。
void FFT(char* input,char* output,int width,int height)	FFT 函数。
void LowPassOrHighPassFiltering(char* input,char* output,int width,int height,int LOW_PASS,int DEGREE)	低通或高通滤波。LOW_PASS=1 为低通滤波，否则为高通滤波，DEGREE 为滤波程度，如 DEGREE=0。
void IFFT(char* input,char* output,int width,int height,int LOW_PASS,int DEGREE)	IFFT 函数。LOW_PASS=1 为低通滤波，否则为高通滤波，DEGREE 为滤波程度，如 DEGREE=0。
void BMPtoRAW(char* inputfile,char* outputfile)	BMP 图像转 RAW 图像。支持 24 为 BMP 图像。
void BMPtoRAW1(char* input,char* output)	BMP 图像转 RAW 图像。支持 24 为 BMP 图像。

BMP 图像处理

unsigned char** BMPRead8(char* input)	读取 8 位 BMP 图像的像素。
void GenerateImage8(char* output,unsigned char** color)	生成 8 位 BMP 图像，output 是生成的图像文件名，color 是像素数据。

BMPMat** BMPRead(char* input)	读取 24 位和 32 位 BMP 图像的像素。
	需要引入以下结构体：
	typedef struct {
	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP

	图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;

unsigned int BMPHeight(char* input)	读取 BMP 图像的高度。
unsigned int BMPWidth(char* input)	读取 BMP 图像的宽度。

void GenerateImage(char* output,BMPMat** color,unsigned short type)	生成 24 位和 32 位 BMP 图像。type 等于图像的位数，如 type=24。
	参考用例：
	BMPMat** color = (BMPMat**)malloc(sizeof(BMPMat*)*1280);
	for (unsigned int i = 0; i < 1280; i++)
	{
	color[i] = (BMPMat*)malloc(sizeof(BMPMat)*2450);
	}
	for (unsigned int i = 0; i < 1280; i++) {
	for (unsigned int j = 0; j < 2450; j++) {
	color[i][j].B =0;
	color[i][j].G =0;
	color[i][j].R =255;
	}
	}

void HistogramEqualization5(char* input,char* output)	直方图均衡，支持 8 位和 16 位 BMP。input 是输入文件名，output 是输出文件名。
void Resize(char* input,char* output,int Height,int Width)	图片缩放，支持 8 位和 16 位 BMP。input 是输入文件名，output 是输出文件名。Height 和 Width 是输出图像的高和宽。
double MeanBrightness(char* input)	求图像的平均亮度，支持 8 位和 16 位 BMP。input 是输入文件名。
int IsBitMap(FILE *fp)	判断是否是位图。
int getWidth(FILE *fp)	获得图片的宽度。
int getHeight(FILE *fp)	获得图片的高度。
unsigned short getBit(FILE *fp)	获得每个像素的位数。

unsigned int getOffSet(FILE *fp)	获得数据的起始位置。
void BMPtoYUV(char* input,char* output, char yuvmode)	BMP 图像转为 YUV 图像，input 是输入文件名，output 是输出文件名。yuvmode 是 YUV 文件的 3 个模式选项，yuvmode 的值可为'0'、'2'、'4'，分别为 420，422，444
void BMPtoYUV420I(char* input,char* output)	BMP 图像转为 YUV420 图像,input 是输入文件名，output 是输出文件名。
void BMPtoYUV420II(char* input,char* output)	BMP 图像转为 YUV420 图像,input 是输入文件名，output 是输出文件名。
void DCMtoBMP(string input,char* output)	DCM 图像转 BMP 图像。input 是输入文件名，output 是输出文件名。
void Ins1977(char* input,char* output,int ratio)	Ins1977 滤镜,input 是输入文件名,output 是输出文件名。参考：ratio=100。
void LOMO(char* input,char* DarkAngleInput,char* output,int ratio)	LOMO 滤镜，DarkAngleInput 是暗角模板图像名，参考：ratio=100。
void PNGGray(char* input,char* output)	图像灰度化，input 是输入文件名，output 是输出文件名。
void PNGSpotlight(char* input,char* output,int centerX,int centerY,double a,double b,double c,double d,double e)	聚光灯效果，input 是输入文件名，output 是输出文件名。焦点坐标（centerX,centerY），如：centerX=400,centerY=180；a、b、c、d、e 是相关参数，默认 a=100，b=100，c=160，d=80，e=0.5。
void PNGIllinify(char* input,char* output)	幻化效果，input 是输入文件名，output 是输出文件名。
void PNGWaterMark(char* input1,char* input2,char* output)	图像加水印，input1 和 input2 的尺寸必须相同。
void Short(char* input,char* output,int a,int b,int c,double d,int depth)	矮化特效。a=1，b=128，c=2，d=0.5，depth=24。支持 24 位 BMP 图像。
void Rise(char* input,char* output,int a,int b,double c,int d,int depth)	增高特效。a=1，b=128，c=0.5，d=2，depth=24。支持 24 位 BMP 图像。
void Short1(char* input,char* output,int a,int b,double c,double d,int depth)	矮小化特效。a=1，b=128，c=0.5，d=0.5，depth=24。支持 24 位 BMP 图像。
void Handstand(char* input,char* output,int a,int b,double c,int depth)	倒立特效。a=1，b=128，c=0.5，depth=24。支持 24 位 BMP 图像。

void Fat(char* input,char* output,int a,int b,double c,int depth)	肥胖特效。a=1，b=128，c=0.5，depth=24。支持 24 位 BMP 图像。
void HighFoot(char* input,char* output,int a,int b,int c,double d,int depth)	高脚特效。a=1，b=128，c=2，d=0.5，depth=24。支持 24 位 BMP 图像。
void CurvedCurve(char* input,char* output,int a,int b,int c,int d,double e,int depth)	弧度弯曲特效。a=1,b=128,c=4,d=2,e=0.5，depth=24。支持 24 位 BMP 图像。
void Thin(char* input,char* output,int a,int b,double c,double d,int depth)	细化特效。a=1，b=128，c=0.5，d=0.5，depth=24。支持 24 位 BMP 图像。
void Winding(char* input,char* output,int lim,int a,int b,int c,int d,double e,int depth)	弯曲特效。lim=20，a=1，b=128，c=4，d=5，e=0.5，depth=24。支持 24 位 BMP 图像。

void CrossDenoising(unsigned char** input,unsigned char** output,double a)	十字法剔除孤立像素点。
	需引入以下结构体和声明：
	typedef struct {
	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP 图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;
	
	typedef struct {
	double B;
	double G;
	double R;
	double A;
	}BMPMatdouble;
	
	void Conversion8(unsigned char** input,double** output);
	void Conversion8(double** input,unsigned char** output);
	void Conversion24(BMPMat** input,BMPMatdouble** output);
	void Conversion24(BMPMatdouble**

	input,BMPMat** output);

void CrossDenoising(BMPMat** input,BMPMat** output,double a)	十字法剔除孤立像素点。
	需引入以下结构体和声明：
	typedef struct {
	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP 图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;
	
	typedef struct {
	double B;
	double G;
	double R;
	double A;
	}BMPMatdouble;
	
	void Conversion8(unsigned char** input,double** output);
	void Conversion8(double** input,unsigned char** output);
	void Conversion24(BMPMat** input,BMPMatdouble** output);
	void Conversion24(BMPMatdouble** input,BMPMat** output);

void CrossConnectionDenoising(unsigned char** input,unsigned char** output,double a)	交叉法剔除孤立像素点。
	需引入以下结构体和声明：
	typedef struct {
	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP 图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;
	
	typedef struct {
	double B;

	double G;
	double R;
	double A;
	}BMPMatdouble;
	
	void Conversion8(unsigned char** input,double** output);
	void Conversion8(double** input,unsigned char** output);
	void Conversion24(BMPMat** input,BMPMatdouble** output);
	void Conversion24(BMPMatdouble** input,BMPMat** output);

void CrossConnectionDenoising(BMPMat** input,BMPMat** output,double a)	交叉法剔除孤立像素点。
	需引入以下结构体和声明：
	typedef struct {
	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP 图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;
	
	typedef struct {
	double B;
	double G;
	double R;
	double A;
	}BMPMatdouble;
	
	void Conversion8(unsigned char** input,double** output);
	void Conversion8(double** input,unsigned char** output);
	void Conversion24(BMPMat** input,BMPMatdouble** output);
	void Conversion24(BMPMatdouble** input,BMPMat** output);

void MatrixDenoising(unsigned char** input,unsigned char**	矩阵法剔除孤立像素点。 需引入以下结构体和声明： typedef struct {

output,double a)	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP 图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;
	
	typedef struct {
	double B;
	double G;
	double R;
	double A;
	}BMPMatdouble;
	
	void Conversion8(unsigned char** input,double** output);
	void Conversion8(double** input,unsigned char** output);
	void Conversion24(BMPMat** input,BMPMatdouble** output);
	void Conversion24(BMPMatdouble** input,BMPMat** output);

void MatrixDenoising(BMPMat** input,BMPMat** output,double a)	矩阵法剔除孤立像素点。
	需引入以下结构体和声明：
	typedef struct {
	unsigned char B; //24 位和 32 位 BMP 图像的蓝色通道分量
	unsigned char G; //24 位和 32 位 BMP 图像的绿色通道分量
	unsigned char R; //24 位和 32 位 BMP 图像的红色通道分量
	unsigned char A; //仅限 32 位 BMP 图像的 Alpha 通道
	}BMPMat;
	
	typedef struct {
	double B;
	double G;
	double R;
	double A;
	}BMPMatdouble;

	void Conversion8(unsigned char** input,double** output);
	void Conversion8(double** input,unsigned char** output);
	void Conversion24(BMPMat** input,BMPMatdouble** output);
	void Conversion24(BMPMatdouble** input,BMPMat** output);

void ImageFusion3(char* input1,char* input2,char* output,int block_height,int block_width,double threshold)	多聚焦图像的融合，支持 8 位 BMP 图像。block_height=8 ， block_width=8 ，threshold=1.75。
void ImageFusion4(char* input1,char* input2,char* output,int block_height,int block_width,double threshold)	多聚焦图像的融合，支持 8 位 BMP 图像。block_height=8 ， block_width=8 ，threshold=1.75。
void ImageFusion5(char* input1,char* input2,char* MaskImage,char* output,int dx[],int dy[],int a,double b1,int DX1,int DY1,double EPS)	图像融合。参考：a=3,b1=4,DX1=-68,DY1=-99，EPS=1，input1="图像融合 1.jpg"，input2="图像融合 2.jpg"，MaskImage="掩膜.png"，output="output.jpg"。 int dx[] = {0,0,-1,1}; int dy[] = {-1,1,0,0};
void Dark(char* input,char* output,int ratio)	暗调滤镜，参考：ratio=100。
void WaveFilter(char* input,char* output,int degree,int a)	波浪形变特效滤镜，degree 是滤镜程度（波浪扭曲度）。a=0 时生成 BMP 图像，a=1 时生成 JPG 图像，a=2 时生成 PNG 图像，a=3 时生成 TGA 图像，参考：degree=10。
void PinchFilter(char* input,char* output,int a)	挤压形变特效滤镜，a=0 时生成 BMP 图像，a=1 时生成 JPG 图像，a=2 时生成 PNG 图像，a=3 时生成 TGA 图像。
void PinchFilter(char* input,char* output,int cenX,int cenY,int a)	挤压形变特效滤镜，a=0 时生成 BMP 图像，a=1 时生成 JPG 图像，a=2 时生成 PNG 图像，a=3 时生成 TGA 图像，cenX 是形变中心点 X 坐标，cenY 是形变中心点 Y 坐标。
void SpherizeFilter(char* input,char* output,int a)	球面形变特效滤镜，a=0 时生成 BMP 图像，a=1 时生成 JPG 图像，a=2 时生成 PNG 图像，a=3 时生成 TGA 图像。
void SpherizeFilter(char* input,char* output,int cenX,int cenY,int a)	球面形变特效滤镜，a=0 时生成 BMP 图像，a=1 时生成 JPG 图像，a=2 时生成 PNG 图像，a=3 时生成 TGA 图像，cenX 是形变中心点 X

	坐标，cenY 是形变中心点 Y 坐标。
void SwirlFilter(char* input,char* output,int ratio,int a)	旋转形变特效滤镜，a=0 时生成 BMP 图像，a=1 时生成 JPG 图像,a=2 时生成 PNG 图像，a=3 时生成 TGA 图像，ratio=3。
void SwirlFilter(char* input,char* output,int cenX,int cenY,int ratio,int a)	旋转形变特效滤镜，a=0 时生成 BMP 图像，a=1 时生成 JPG 图像,a=2 时生成 PNG 图像，a=3 时生成 TGA 图像，ratio=3，cenX 是形变中心点 X 坐标，cenY 是形变中心点 Y 坐标。
void ClosedOperation(char* input,char* output)	闭运算，input 是输入文件名，output 是输出文件名。支持 4 位 BMP 图像。
void AdjustPixel(char* input,char* output,int a)	调整像素值，input 是输入文件名，output 是输出文件名。a 是用于设置图像像素的相关参数，如 a=3。支持 24 位 BMP 图像。
void PartialColorRetention(char* input,char* output,int ratio)	部分颜色保留滤镜，参考：ratio=60。
void GrayImageConversion8(char* input,char* output)	生成图像的灰度图，支持 8 位 BMP 图像。input 是输入文件名，output 是输出文件名。
void Gray(char* input,char* output)	灰度图转换，支持 24 位 BMP 图像。input 是输入文件名，output 是输出文件名。
void GrayImageConversion(char* input,char* output)	彩色图转灰度图，input 是要处理的彩色图像，output 是处理后生成的灰度图名称。支持 24 位 BMP 图像。
void BinaryImageVerticalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	二值图像垂直镜像，input 是输入图像的像素数据，output 是输出图像的像素数据，w 是输入图像的宽，h 是输入图像的高。
void GrayImageVerticalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	灰度图像垂直镜像，input 是输入图像的像素数据，output 是输出图像的像素数据，w 是输入图像的宽，h 是输入图像的高。
void ColorImageVerticalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	彩色图像垂直镜像，input 是输入图像的像素数据，output 是输出图像的像素数据，w 是输入图像的宽，h 是输入图像的高。
void OTSU(char* input,char* output,int BeforeThreshold)	大津算法，input 是输入文件名，output 是输出文件名。BeforeThreshold 是初始阈值，如 BeforeThreshold=10。支持 8 位 BMP

	图像。
void LowerBrightness(char* input,char* output,int a,int b)	调低亮度，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。a 和 b 的参考值可为 a=100，b=0。
void HightBrightness(char* input,char* output,int a,int b)	调高亮度，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。a 和 b 的参考值可为 a=100，b=0。
void IterativeThresholdSelection(char* input,char* output)	迭代阈值选择，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void DitheringMethod(char* input,char* output)	抖动法，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void LogTransformation(char* input,char* output,int constant)	对数变换，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。constant 是相关参数，如 constant=15。
void LogarithmicTransformation(char* input,char* output)	对数变换，input 是输入文件名，output 是输出文件名。支持 BMP 图像。
void HistogramEqualization(char* input,char* output)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 BMP 图像。
void Binarization(char* input,char* output,int threshold)	二值化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。threshold 是阈值，如：threshold=128。
void Expansion(char* input,char* output,unsigned char mask[9],int c)	二值图像膨胀，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void Corrosion(char* input,char* output,unsigned char mask[9],int c)	二值图像腐蚀，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void OpenOperation(char* input,char* output,unsigned char mask[9],int c)	二值图像开运算，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void ClosedOperation(char* input,char* output,unsigned char mask[9],int c)	二值图像闭运算，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void OpenOperationToExtractContour(char* input,char* output,unsigned char mask[9],int c)	二值图像开运算提取轮廓，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void ExpansionOperationToContourExtraction(char* input,char*	二值图像膨胀运算提取轮廓，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。

output,unsigned char mask[9],int c)	
void CorrosionCalculationToContourExtraction(char* input,char* output,unsigned char mask[9],int c)	二值图像腐蚀运算提取轮廓，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void Glaw(char* input,char* output,int ratio)	发光滤镜，参考：ratio=100。
void LowPassFilter(char* input,char* output)	低通滤波器，input 是输入文件名，output 是输出文件名。支持 BMP 图像。
void HighPassFilter(char* input,char* output)	高通滤波器，input 是输入文件名，output 是输出文件名。支持 BMP 图像。
void Thinning(char* input,char* output)	图像细化，input 是输入文件名，output 是输出文件名。支持 BMP 图像。
void ThinningLine(char* input,char* output)	图像细化且线条化，input 是输入文件名，output 是输出文件名。支持 BMP 图像。
void Corrosion(char* input,char* output)	腐蚀，input 是输入文件名，output 是输出文件名。支持 4 位 BMP 图像。
void Corrosion1(char* input,char* output,int *TempBuf, int TempH, int TempW)	腐蚀，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。TempBuf 是腐蚀模板，TempH 和 TempW 分别是 TempBuf 的高和宽，如 TempH=4,TempW=4，则有 TempBuf[4][4]。
void Expand(char* input,char* output,int *TempBuf, int TempH, int TempW)	膨胀，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。TempBuf 是膨胀模板，TempH 和 TempW 分别是 TempBuf 的高和宽，如 TempH=4,TempW=4，则有 TempBuf[4][4]。
unsigned char** create2DImg(unsigned char* input, int w, int h)	线性存储的灰阶图像像素转化为二维。
unsigned char getMaxPixelWhole(unsigned char **input,int x,int y,int w,int h,int *Kernal,int kernalW,int halfKernalW)	图像指定区域取最大值（判断是否超出边界）。
unsigned char getMaxPixelCenter(unsigned char **input,int x,int y,int *Kernal,int kernalW,int halfKernalW)	图像指定区域取最大值(不判断是否超出边界)。
unsigned char** imgDilate(unsigned char	图像膨胀。

*input,int w,int h,int *Kernal,int kernalW,int halfKernalW)	
unsigned char getMinPixelWhole(unsigned char **input,int x,int y,int w,int h,int *Kernal,int kernalW,int halfKernalW)	图像指定区域取最小值（判断是否超出边界）。
unsigned char getMinPixelCenter(unsigned char **input,int x,int y,int *Kernal,int kernalW,int halfKernalW)	图像指定区域取最小值(不判断是否超出边界)。
unsigned char** imgErode(unsigned char *input,int w,int h,int *Kernal,int kernalW,int halfKernalW)	图像腐蚀。
void Corrosion(unsigned char *input,unsigned char *output,int rows,int cols,int mat[5][5])	二值腐蚀。
void Expansion(unsigned char *input,unsigned char *output,int rows,int cols,int mat[5][5])	二值膨胀。
void BoxBlurAdvanced(string input,string output,int radius)	高级方框模糊，参考：radius=5。支持 PNG 文件。
void GaussianBlurFilter(char* input,char* output)	高斯滤波，支持 PNG 文件。
void GaussianFiltering(char* input,char* output)	高斯滤波，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void LaplaceEnhancement(char* input,char* output)	拉普拉斯增强，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void Residual(char* input,char* output)	求残差，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void SunlightFilter(char* input,char* output,int intensity,int radius,int x,int y)	光照特效滤镜，intensity 是光照强度，如：intensity=255；radius 是光照范围，如：radius=600；x 和 y 是光照的位置，如：x=100，y=60。
void Compress(char*	压缩，支持多种文件。input 是要压缩的文

input,char* output)	件名，output 是压缩后的文件名。
void Decompression(char* input, char* output)	解压缩，支持多种文件。input 是要解压缩的文件名，output 是解压缩后的文件名。
void BlackWhite(char* input,char* output,int width,int height,unsigned char threshold1,unsigned char threshold2,unsigned char threshold3,unsigned char color1,unsigned char color2)	黑白化，参考：threshold1、threshold2 和 threshold3 都等于 128，color1=255，color2=0。支持 24 位 BMP 图像。
void BlackWhite(char* input,char* output)	黑白化，input 是输入的原图像，output 是输出的黑白图像。支持 24 位 BMP 图像。
void Underexposure(char* input,char* output)	图像欠曝光，input 是输入的原图像，output 是输出的欠曝光图像。支持 24 位 BMP 图像。
void Overexposure(char* input,char* output)	图像过曝光，input 是输入的原图像，output 是输出的过曝光图像。支持 24 位 BMP 图像。
void Nostalgia(char* input,char* Mask,char* output,int ratio)	怀旧滤镜，input 和 Mask 都是输入的文件名，Mask 是褶皱图像路径，ratio=100。
void GammaTransform(char* input,char* output)	伽马变换，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void GrayScale(char* input,char* output)	灰度化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void GrayImageBinarization(char* input,char* output,int bit,int threshold)	灰度图二值化，bit 用于设定位数，如 bit=8；threshold 是阈值，如 threshold=200。支持 8 位 BMP 图像。
void GreyPesudoColor(char* input,char* output)	灰度图伪彩色化，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void HoughTransform(char* input,char* output,unsigned char threshold)	霍夫变换，input 是输入的 RAW 文件，output 是输出的 RAS 文件，threshold=100。
static void EdgeDetectionWithoutNonmaximum(const LPCTSTR input, const LPCTSTR output,double a,double b,double c)	边缘检测，参考：a=0.33，b=0.33，c=0.33。支持 24 位 BMP 图像。
static void NonmaximumWithoutDoubleThresholding(const LPCTSTR input, const LPCTSTR output,double	参考：a=0.33，b=0.33，c=0.33。支持 24 位 BMP 图像。

a,double b,double c)	
static void CannyEdgeDetection(const LPCTSTR input, const LPCTSTR output,double a,double b,double c,int orank,int oranb)	边缘检测，参考：orank=20，oranb=80。支持 24 位 BMP 图像。
static void HoughTransform(const LPCTSTR input, const LPCTSTR output,double a,double b,double c,int orank,int oranb)	霍夫变换，参考：a=0.33，b=0.33，c=0.33，orank=20，oranb=80。支持 24 位 BMP 图像。
void BoxBlurBasic(string input,string output)	基础方框模糊，支持 PNG 文件。
void CalculateCumulativeHistogramMap(char* input,char* outfile)	计算累加直方图并映射，input 是输入文件名，outfile 是输出文件名。支持 24 位 BMP 图像。
void Translation(string input,char* output,int dx,int dy)	图像平移，input 是输入的文件，dx 和 dy 是横向及纵向的移动距离（像素），负值是向左 / 向下移动；output 是平移操作后的结果文件名。支持 BMP 图像。
void Mirrored(string input,char* output,char axis)	镜像变换，input 是输入的文件，output 是镜像操作后的结果文件名，axis 是镜像变换的方向（以 X 或 Y 表示）。支持 BMP 图像。
void Sheared(string input,char* output,char axis,double Coef)	错切变换，input 是输入的文件，output 是错切操作后的结果文件名，axis 和 Coef 分别是错切变换的方向（以 X 或 Y 表示）和错切系数，负值是向左 / 向下偏移。支持 BMP 图像。
void Scaled(string input,char* output,double cx,double cy)	缩放操作，input 是输入的文件，output 是缩放操作后的结果文件名，cx 和 cy 分别是横向及纵向的缩放系数，系数大于 1 表示拉伸，小于 1 表示压缩。支持 BMP 图像。
void Rotated1(string input,char* output,double angle)	图像旋转，input 是输入的文件，output 是图像旋转后的结果文件名，angle 是旋转角度，弧度制。支持 BMP 图像。
void SaltNoise(char* input,char* output,int a,int b,int c,int d)	添加椒盐噪声，a 和 b 是噪声相关参数，如 a=3，b=3；c 和 d 是颜色相关参数，如 c=0，d=255。支持 8 位 BMP 图像。
void CrossProcess(char* input,char* output,int ratio)	交叉冲印滤镜，参考：ratio=100。

void Conversion8(unsigned char** input,short** output)	unsigned char**转 short**，output 用于保存结果（与 input 大小相同）。
void Conversion8(short** input,unsigned char** output)	short**转 unsigned char**，output 用于保存结果（与 input 大小相同）。
void Conversion8(unsigned char** input,int** output)	unsigned char**转 int**，output 用于保存结果（与 input 大小相同）。
void Conversion8(int** input,unsigned char** output)	int**转 unsigned char**，output 用于保存结果（与 input 大小相同）。
void Conversion8(unsigned char** input,unsigned int** output)	unsigned char**转 unsigned int**，output 用于保存结果(与 input 大小相同)。
void Conversion8(unsigned int** input,unsigned char** output)	unsigned int**转 unsigned char**，output 用于保存结果(与 input 大小相同)。
void Conversion8(unsigned char** input,float** output)	unsigned char**转 float **，output 用于保存结果（与 input 大小相同）。
void Conversion8(float** input,unsigned char** output)	float **转 unsigned char**，output 用于保存结果（与 input 大小相同）。
void Conversion8(unsigned char** input,double** output)	unsigned char**转 double **，output 用于保存结果（与 input 大小相同）。
void Conversion8(double** input,unsigned char** output)	double **转 unsigned char**，output 用于保存结果（与 input 大小相同）。
void Conversion8(unsigned char** input,char** output)	unsigned char**转 char **，output 用于保存结果（与 input 大小相同）。
void Conversion8(char** input,unsigned char** output)	char **转 unsigned char**，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMat** input,BMPMatshort** output)	BMPMat **转 BMPMatshort **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMatshort** input,BMPMat** output)	BMPMatshort **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMat** input,BMPMatint** output)	BMPMat **转 BMPMatint **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMatint** input,BMPMat** output)	BMPMatint **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMat** input,BMPMatfloat** output)	BMPMat **转 BMPMatfloat **，output 用于保存结果（与 input 大小相同）。

void Conversion24(BMPMatfloat** input,BMPMat** output)	BMPMatfloat **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMat** input,BMPMatdouble** output)	BMPMat **转 BMPMatdouble **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMatdouble** input,BMPMat** output)	BMPMatdouble **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMat** input,BMPMatchar** output)	BMPMat **转 BMPMatchar **，output 用于保存结果（与 input 大小相同）。
void Conversion24(BMPMatchar** input,BMPMat** output)	BMPMatchar **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMat** input,BMPMatshort** output)	BMPMat **转 BMPMatshort **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMatshort** input,BMPMat** output)	BMPMatshort **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMat** input,BMPMatint** output)	BMPMat **转 BMPMatint **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMatint** input,BMPMat** output)	BMPMatint **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMat** input,BMPMatfloat** output)	BMPMat **转 BMPMatfloat **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMatfloat** input,BMPMat** output)	BMPMatfloat **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMat** input,BMPMatdouble** output)	BMPMat **转 BMPMatdouble **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMatdouble** input,BMPMat** output)	BMPMatdouble **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMat** input,BMPMatchar** output)	BMPMat **转 BMPMatchar **，output 用于保存结果（与 input 大小相同）。
void Conversion32(BMPMatchar** input,BMPMat** output)	BMPMatchar **转 BMPMat **，output 用于保存结果（与 input 大小相同）。
void MeanFiltering(char* input,char* output)	均值滤波，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void MeanFltering1(char* input,char* output)	均值滤波，input 是输入文件名，output 是输出文件名。支持 8 位和 24 位 BMP 图像。
void KapoorAlgorithm(char* input,char* output,int	卡普尔算法，input 是输入文件名，output 是输出文件名。BeforeThreshold 是初始阈

BeforeThreshold)	值,如 BeforeThreshold=150。支持 8 位 BMP 图像。
void OpenOperation(char* input,char* output)	开运算，input 是输入文件名，output 是输出文件名。支持 4 位 BMP 图像。
void Diffusion(char* input,char* output,int ratio)	扩散滤镜，参考：ratio=90。
void LapulasFiltering(char* readPath,char* writePath,float CoefArray[9],float coef)	拉普拉斯滤波，readPath 是原图像，writePath 是处理后的图像文件名。支持 8 位 BMP 图像。 各参数参考值： 定义*3 的模板（拉普拉斯）： float CoefArray[9]={1.0f,2.0f,1.0f,2.0f,4.0f,2.0f,1.0f,2.0f,1.0f}; 定义模板前乘的系数（拉普拉斯）： float coef=(float)(1.0/16.0);
void ImageFiltering(char* input,char* output,float kernel[3][3])	图像滤波，input 是输入文件名，output 是输出文件名。kernel 是模糊内核。支持 24 位 BMP 图像。
void ComicStrip(char* input,char* output,int ratio)	连环画滤镜，参考：ratio=100。
void BrightnessAdjustment1(char* input,char* output,int brightness,int contrast)	亮度对比度调节，参考：brightness=-30，contrast=100。
void BrightnessAdjustment2(char* input,char* output,int brightness,int contrast)	亮度对比度调节，参考：brightness=-30，contrast=100。
void ZeroFillingSymmetricExtension(char* input,char* output)	零填充与对称扩展，支持 8 位和 24 位 BMP 图像。
void PopArtStyle(char* input,char* output,int ratio)	流行艺术风滤镜，参考：ratio=100。
void LightLeakage(char* input,char* Mask,char* output,int ratio)	漏光滤镜，input 和 Mask 都是输入的图像名，Mask 是漏光模板图像，ratio=90。
void LinearFiltering(char* input,char* output,short average[3][3])	线性滤波，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。 参考模板： short average[3][3] = {{1, 2, 1},

	{2, 4, 2}, {1, 2, 1}}; short sharpen[3][3] = {{-1, -1, -1}, {-1, 8, -1}, {-1, -1, -1}};
void MedianFilering(char* input,char* output,short average[3][3])	中值滤波，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。 参考模板： short average[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
void SharpeningFiltering(char* input,char* output,short average[3][3],short sharpen[3][3])	锐化滤波，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。 参考模板： short average[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}}; short sharpen[3][3] = {{-1, -1, -1}, {-1, 8, -1}, {-1, -1, -1}};
void GradientSharpening(char* input,char* output,short average[3][3],short soble1[3][3],short soble2[3][3])	梯度锐化，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。 参考模板： short average[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}}; short soble1[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}}; short soble2[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
void ArithmeticMeanFilter(char* input,char* output)	算术平均滤波器，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void GeometricMeanFilter(char* input,char* output)	几何平均滤波器，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void HarmonicMeanFilter(char* input,char* output)	调和平均滤波器，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void ContraHarmonicMeanFilter(char* input,char* output)	反调和平均滤波器，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Filter(char* input,char* output)	滤波，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void GreenAtmosphere(char*	绿色氛围。

input,char* output)	
void GreenAtmosphere(char* input,char* output,int width,int height,double a)	绿色氛围。支持 24 位 BMP 图像。
void BMPtoJPG(char* input,char* output,int a)	BMP 图像转为 JPG 图像。支持 24 位 BMP 图像，尺寸必须是 8 的倍数，a 代表文件压缩程度，数字越大，压缩后的文件体积越小，如 a=100。
void Mosaic(char* input,char* output,int x)	马赛克化图像，input 是输入文件名，output 是输出文件名。x 是马赛克处理的块的大小。支持 24 位 BMP 图像。
void MosaicFilter(char* input,char* output,int ratio)	马赛克滤镜，参考：ratio=50。
void MuddyAtmosphere(char* input,char* output)	泥色氛围。
void Expansion(char* input,char* output)	膨胀，input 是输入文件名，output 是输出文件名。支持 4 位 BMP 图像。
void SmoothSharpen(char* input,char* output,int Template[3][3],int coefficient)	平滑，input 是输入文件名，output 是输出文件名。Template 是平滑模板，均一化处理，coefficient1 = 9。支持 24 位 BMP 图像。
void GaussSmoothSharpen(char* input,char* output,int Template[3][3],int coefficient)	高斯平滑，input 是输入文件名，output 是输出文件名。Template 是高斯平滑模板，coefficient=16。支持 24 位 BMP 图像。
void SobelSharpen(char* input,char* output,int Templatex[3][3],int Templatey[3][3],int coefficient1,int coefficient2)	Sobel 算子，input 是输入文件名，output 是输出文件名。Templatex 是 laplace 锐化模板,4 邻域,Templatey 是 laplace 锐化模板 ,8 邻域，coefficient1 = 9，coefficient2 = 16。支持 24 位 BMP 图像。
void MidSmoothing(char* input,char* output)	中值滤波器，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void AvgSmoothing(char* input,char* output)	均值滤波器，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Averaging(char* input1,char* input2,char* input3,char* output,int a)	图像平均化，input 是输入文件名，output 是输出文件名。a 是平均化相关参数，如 a=3。支持 8 位 BMP 图像。
void PlaneSlicing(char* input,char* output)	平面切片，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Translation(char* input,char* output,int	图像平移，参考：xoffset=-100，yoffset=-100。

xoffset,int yoffset)	
void SharpeningSpatialFiltering8(char* input,char* output,int model[9])	锐化空间滤波器，input 是输入文件名，output 是输出文件名。model 是锐化模板。支持 8 位灰度图像。
void PseudoGrayscale(char* input,char* output)	伪灰度化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void TwoColors(char* input,char* output,int threshold,unsigned char color1,unsigned char color2)	二色化，input 是输入文件名，output 是输出文件名。threshold 是阈值，如 threshold=115；color1 和 color2 是要填充的两个颜色。支持 24 位 BMP 图像。
void PNGImageGeneration(char* filename,const unsigned char img[],unsigned W,unsigned H,int x)	filename 是生成的 PNG 图像文件名;img 是图像的像素数据，W 是图像的宽，H 是图像的高，x=0 选择生成 RGB 图像，x=1 选择生成 RGBA 图像。
void MakeSphere(double V[3],double S[3], double r, double a, double m, int ROWS, int COLS, char* output)	使用反射模型在正交投影下生成球体的图像，V 是摄影机的方向，output 是输出的结果图像文件名，ROWS 是输出图像的行数，COLS 是输出图像的列数，参考：V[3] = {0.0, 0.0, 1.0}，S[3] = {0.0, 0.0, 1.0}，r=50，a=0.5，m=1。支持 RAS 文件。
void MakeSphere(double vector_v[3],double vector_s[3], double r, double a, double m, int ROWS,int COLS,char* output)	使用反射模型生成球体的图像，vector_v 是摄影机的方向，output 是输出的结果图像文件名，ROWS 是输出图像的行数，COLS 是输出图像的列数，参考：vector_v[3] = {0.0, 0.0, 1.0}，vector_s[3] = {0.0, 0.0, 1.0}，r=50，a=0.5，m=1。支持 RAS 文件。
void BilateralFiltering(string input,char* output,double ssd, double sdid)	双边滤波，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。ssd 和 sdid 分别是空间域标准差与强度域标准差。
void DoubleLayerErosion(char* input,char* output)	具有圆形结构集的双层形态侵蚀,支持 8 位和 24 位 BMP 图像。
void BinaryImageHorizontalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	二值图像水平镜像。
void GrayImageHorizontalMirror(un	灰阶图像水平镜像。

signed char *input,unsigned char *output,unsigned int w,unsigned int h)	
void ColorImageHorizontalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	彩色图像水平镜像。
void SketchFilter(char* input,char* output,int ratio)	素描滤镜，参考：ratio=100。
void Zoom(char* input,char* output,float scaleX,float scaleY,int interpolation)	缩放，参考：scaleX=5，scaleY=5，interpolation=0 或 interpolation=1。
int Equal(char* input1,char* input2,double c)	若比对图像的梯度幅相似性偏差值等于 c 则通过。input1 和 input2 是要比对的两个图像。c 是参考的阈值。支持 24 位 BMP 图像。
int GreaterThan(char* input1,char* input2,double c)	若比对图像的梯度幅相似性偏差值大于 c 则通过。input1 和 input2 是要比对的两个图像。c 是参考的阈值。支持 24 位 BMP 图像。
int LessThan(char* input1,char* input2,double c)	若比对图像的梯度幅相似性偏差值小于 c 则通过。input1 和 input2 是要比对的两个图像。c 是参考的阈值。支持 24 位 BMP 图像。
double GMSD(char* input1, char* input2)	求两幅图像的梯度幅相似性偏差值并返回结果。input1 和 input2 是要比对的两个图像。支持 24 位 BMP 图像。
void AddGaussNoise(char* input,char* output)	添加高斯噪声，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void AddSaltPepperNoise(char* input,char* output)	添加椒盐噪声，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void ChannelSeparation(char* input,char* Routput,char* Goutput,char* Boutput)	通道分离，input 是输入文件名，Routput 是红色通道图像，Goutput 是绿色通道图像，Boutput 是绿色通道图像。支持 24 位 BMP 图像。
void PatternMethod(char* input,char* output,unsigned char Template[8][8])	图案法，input 是输入文件名，output 是输出文件名。Template 是模板数组。支持 8 位 BMP 图像。
void LayerAlgorithm(char*input,char* inputMix,char*	图层算法，input 是基底图层图像，inputMix 是混合图层图像，参考：alpha=50，blendMode1=26。

output,int alpha,int blendModel)	blendModel 的取值对应的模式如下： 1 典型 2 溶解 3 暗化 4 多层 5 颜色加深模式 6 线性加深 7 暗调 8 亮化 9 遮盖 10 颜色减淡模式 11 线性减淡 12 浅色 13 叠加 14 柔光模式 15 强光模式 16 艳光模式 17 线性光模式 18 点光模式 19 强混合模式 20 差分 21 排除模式 22 减运算 23 图像分割 24 色相模式 25 色饱和 26 着色 27 亮度模式
void BMP24LossyCompression(char* input,char* output)	图像有损压缩，input 是待压缩的 BMP 文件名，output 是有损压缩后输出的文件名。支持 24 位 BMP 图像。
void BMP24LossyDecompression(char* input,char* output)	图像有损解压，input 是待解压的文件名，output 是输出解压后的 BMP 文件名。支持 24 位 BMP 图像。
void BMP24LosslessCompression(char* input,char* output)	图像无损压缩，input 是待压缩的 BMP 文件名，output 是无损压缩后输出的文件名。支持 24 位 BMP 图像。
void BMP24LosslessDecompression(char* input,char* output)	图像无损解压，input 是待解压的文件名，output 是输出解压后的 BMP 文件名。支持 24 位 BMP 图像。
void ImageDiscoloration(char* input,char* output,double a,double b,double c)	图像变色，input 是输入文件名，output 是输出文件名。如：a=0.2126，b=0.7152，c=0.0722。支持 24 位 BMP 图像。

unsigned char** HorizontalConcavity(unsigned char** input,int RANGE,int height,int width)	图像变形之水平内凹,返回处理结果。参考：RANGE=400。
unsigned char** HorizontalConvexity(unsigned char** input,int RANGE,int height,int width)	图像变形之水平外凸,返回处理结果。参考：RANGE=400。
unsigned char** TrapezoidalDeformation(unsigned char** input,int height,int width,double k)	图像变形之梯形形变,返回处理结果。参考：k=0.3。
unsigned char** TriangularDeformation(unsigned char** input,int height,int width,double k)	图像变形之三角形形变，返回处理结果。参考：k=0.5。
unsigned char** SDeformation(unsigned char** input,int height,int width,int RANGE)	图像变形之 S 形变，返回处理结果。参考：RANGE=450。
int LsdLineDetector(unsigned char *src, int w, int h,float scaleX, float scaleY, boundingbox_t bbox, std::vector<line_float_t> &lines)	LSD 直线检测器。 [in] src: 图像，单通道 [in] w: 宽 [in] h: 高 [in] scaleX: X 轴上的缩小因子 [in] scaleY: Y 轴上的缩小因子 [in] bbox: 要检测的边界框 [in/out] lines: 结果 return: 0:ok; 1:error 需引入以下结构体： typedef struct { int x; int y; int width; int height; }boundingbox_t; typedef struct { float startx; float starty; float endx; float endy; }line_float_t;

int EdgeDrawingLineDetector(unsigned char *src, int w, int h,float scaleX, float scaleY, boundingbox_t bbox, std::vector<line_float_t> &lines)	边缘划线检测器。 [in] src: 图像，单通道 [in] w: 宽 [in] h: 高 [in] scaleX: X 轴上的缩小因子 [in] scaleY: Y 轴上的缩小因子 [in] bbox: 要检测的边界框 [in/out] lines: 结果 return: 0:ok; 1:error 需引入以下结构体： typedef struct { int x; int y; int width; int height; }boundingbox_t; typedef struct { float startx; float starty; float endx; float endy; }line_float_t;
int PropagatedFilter1(unsigned char *src, unsigned char *guidance, unsigned char *dst,int w, int h, int c, int r, float sigma_s, float sigma_r)	传播滤波器。 [in] src: 输入图像 [in] guidance: 引导图像 [in/out]] dst: 输出图像 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 或 c=3 [in] r: 局部窗口半径 [in] sigma_s:坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor）。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r:颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 @return: 0:ok; 1:error
int	传播滤波器。

PropagatedFilter2(unsigned char *src, unsigned char *guidance, unsigned char *dst,int w, int h, int c, int r, float sigma_s, float sigma_r)	[in] src: 输入图像 [in] guidance: 引导图像 [in/out]] dst: 输出图像 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 或 c=3 [in] r: 局部窗口半径 [in] sigma_s:坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor）。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r:颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 @return: 0:ok; 1:error
int BoxfilterFilter(unsigned char *src, unsigned char *dst,int w, int h, int c, int r)	方盒滤波。 [in] src: 输入图像，单通道 [in/out] dst: 输出图像，单通道 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 [in] r: 局部窗口半径 return: 0:ok; 1:error
int BoxfilterFilter1(unsigned char *src, unsigned char *dst,int w, int h, int c, int r)	方盒滤波。 [in] src: 输入图像，单通道 [in/out] dst: 输出图像，单通道 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 [in] r: 局部窗口半径 return: 0:ok; 1:error
int fast_guided_filter(unsigned char *src, unsigned char *guidance, unsigned char *dst,int w, int h, int c, int r, float rp, float sr,float _scale)	快速导向滤波 [in] src: 输入图像，单通道 [in] guidance: 引导图像，单通道 [in/out] dst: 输出图像，单通道 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 [in] r: 局部窗口半径 [in] rp: 正则化参数：eps [in] sr: 二次采样率，sr>1：向下

	缩放，0<sr<1：向上缩放 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255 return： 0:ok; 1:error eg: r = 4, (try sr = r/4 to sr=r),(try rp=0.1^2, 0.2^2, 0.4^2) try:(src,guidance,dst,w,h,1,4,0.01,4,255*255) condition: (MIN(w, h) / sr) > 1 condition: (int)(r / sr + 0.5f) >= 1
int fast_guided_filter1(unsigned char *src, unsigned char *guidance, unsigned char *dst,int w, int h, int c, int r, float rp, float sr,float _scale)	快速导向滤波 [in] src: 输入图像，单通道 [in] guidance: 引导图像，单通道 [in/out] dst: 输出图像，单通道 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 [in] r: 局部窗口半径 [in] rp: 正则化参数：eps [in] sr: 二次采样率，sr>1：向下缩放，0<sr<1：向上缩放 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255 return： 0:ok; 1:error eg: r = 4, (try sr = r/4 to sr=r),(try rp=0.1^2, 0.2^2, 0.4^2) try:(src,guidance,dst,w,h,1,4,0.01,4,255*255) condition: (MIN(w, h) / sr) > 1 condition: (int)(r / sr + 0.5f) >= 1
int HoughLineDetector(unsigned char *src, int w, int h,float scaleX, float scaleY, float CannyLowThresh, float CannyHighThresh,float HoughRho, float HoughTheta, float MinThetaLinelength, float MaxThetaGap, int HoughThresh,HOUGH_LINE_TYPE_CODE _type,boundingbox_t bbox, std::vector<line_float_t> &lines)	霍夫线探测器。 [in] src: 图像，单通道 [in] w: 宽 [in] h: 高 [in] scaleX: X 轴上的缩小因子 [in] scaleY: Y 轴上的缩小因子 [in] CannyLowThresh: canny 算子中迟滞过程的低阈值 [in] CannyHighThresh: canny 算子中迟滞过程的高阈值 [in] HoughRho: 累加器的距离分辨率（以像素为单位） [in] HoughTheta: 累加器的角度分辨率（弧度）

	[in] MinThetaLinelength: 标准：对于标准和多尺度 hough 变换，检查线条的最小角度
	
	传播能力：最小线路长度。小于的线段被拒绝
	[in] MaxThetaGap: 标准：对于标准和多尺度 hough 变换，检查线条的最大角度
	
	基于概率的：连接同一条线上的点之间允许的最大间隙
	[in] HoughThresh: 累加器阈值参数。只有那些获得足够选票的行才会返回（>阈值）
	[in] _type: hough 线方法：hough_line_STANDARD 或 hough_line_PROBABILISTIC
	[in] bbox: 要检测的边界框
	[in/out] lines: 结果
	return 0:ok; 1:error
	_type: HOUGH_LINE_STANDARD: 标准 hough 线算法
	HOUGH_LINE_PROBABILISTIC： 概率 hough 线算法
	
	当 HOUGH_LINE_STANDARD 运行时，线点可能是图像坐标之外的位置
	
	标准: try (src,w,h,scalex,scaley,70,150, 1, PI/180, 0, PI, 100, HOUGH_LINE_STANDARD, bbox, line)
	基于概率的：try (src,w,h,scalex,scaley,70,150, 1, PI/180, 30, 10, 80, HOUGH_LINE_STANDARD, bbox, line)。
	需引入以下结构体：
	typedef enum _HOUGH_LINE_TYPE_CODE
	{
	HOUGH_LINE_STANDARD = 0,
	//standad hough line
	HOUGH_LINE_PROBABILISTIC = 1,
	//probabilistic hough line
	
	}HOUGH_LINE_TYPE_CODE;

	typedef struct { int x; int y; int width; int height; }boundingbox_t; typedef struct { float startx; float starty; float endx; float endy; }line_float_t;
void _fast_bilateral_filter_singlechannel(unsigned char *src, unsigned char *guidance, unsigned char *dst, int w, int h, float sigma_s, float sigma_r,float _scale)	快速双边滤波器单通道。 [in] src: 输入图像，单通道 [in] guidance: 引导图像，单通道 [in/out] dst: 输出图像，单通道 [in] w: 宽 [in] h: 高 [in] sigma_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor）。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255 return: 0:ok; 1:error
int fast_bilateral_filter_singlechannel(unsigned char *src, unsigned char *guidance, unsigned char *dst, int w, int h, int c, float sigma_s, float sigma_r,float _scale)	快速双边滤波器单通道。 [in] src: 输入图像，单通道 [in] guidance: 引导图像，单通道 [in/out] dst: 输出图像，单通道 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 [in] sigma_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor）。当 d>0 时，它指定邻域大小，

	而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r: 颜色空间中的滤波器西格玛。该参数的值越大,意味着像素邻域(请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255 return： 0:ok; 1:error
void _fast_bilateral_filter_color (unsigned char *src, unsigned char *dst, int w, int h, float sigma_s, float sigma_r,float _scale)	快速双边滤波器 RGB 通道。 [in] src: 输入图像，RGB 通道 [in/out] dst: 输出图像，RGB 通道 [in] w: 宽 [in] h: 高 [in] sigma_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor)。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r: 颜色空间中的滤波器西格玛。该参数的值越大,意味着像素邻域(请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255 return： 0:ok; 1:error
int fast_bilateral_filter_color(unsigned char *src, unsigned char *dst, int w, int h, int c, float sigma_s, float sigma_r,float _scale)	快速双边滤波器 RGB 通道。 [in] src: 输入图像，RGB 通道 [in/out] dst: 输出图像，RGB 通道 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=3 [in] sigma_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor)。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r: 颜色空间中的滤波器西格玛。该参数的值越大,意味着像素邻域(请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 如果正则化，_scale = 1; 如果不正则化，

	_scale = 255*255 return： 0:ok; 1:error
int FastBilateralFilter(unsigned char *src, unsigned char *guidance, unsigned char *dst, int w, int h, int c, float sigma_s, float sigma_r,float _scale)	快速双边滤波器。 [in] src: 输入图像 [in] guidance: 引导图像，单个通道，只有单个通道有效 [in/out] dst: 输出图像 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 或 c=3 [in] sigma_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor）。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255 return： 0:ok; 1:error 如果引导为 NULL，仍然可以获得滤色器
int permutohedral_bilateral_filter(unsigned char *src, unsigned char *guidance, unsigned char *dst,int w, int h, int c, float sigma_s, float sigma_r,float _scale)	快速双边滤波器。 [in] src: 输入图像 [in] guidance: 引导图像 [in/out] dst: 输出图像 [in] w: 宽 [in] h: 高 [in] c: 图像通道，仅 c=1 或 c=3 [in] sigma_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见 sigmaColor）。当 d>0 时，它指定邻域大小，而不考虑 sigmaSpace。否则，d 与 sigmaSpace 成比例。 [in] sigma_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见 sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。 如果正则化，_scale = 1; 如果不正则化，_scale = 255*255

	return: 0:ok; 1:error try:(src,guidance,dst,w,h,c,1.6f,0.6f,255*255)
void HighPassFilter(char* input,char* output,int preserve)	高通滤波器，参考：preserve=0。
void EmbossFilter(char* input,char* output,int preserve)	浮雕过滤器，参考：preserve=1。
void SharpenFilter(char* input,char* output,int preserve)	锐化过滤器，参考：preserve=1。
void Convolution(char* input,char* output,int w,int preserve)	卷积，参考：w=7，preserve=1。
void GaussianBlur(char* input,char* output,float sigma,int preserve)	高斯模糊，参考：sigma=2，preserve=1。
void HybridImage(char* input1,char* input2,char* output,float sigma,int preserve)	混合图像，参考：sigma=2，preserve=1。
void LowFrequencyImage(char* input,char* output,float sigma,int preserve)	低频图像，参考：sigma=2，preserve=1。
void HighFrequencyImage(char* input,char* output,float sigma,int preserve)	高频图像，参考：sigma=2，preserve=1。
void HighFrequencyImage1(char* input,char* output,float sigma,int preserve)	高频图像，参考：sigma=2，preserve=1。
void Bilateral(char* input,char* output,float sigma1, float sigma2)	双边滤波，参考：sigma1=3，sigma2=0.1。
void SkinSmooth(char* input,char* output,int a,int b)	皮肤细滑，a 是平滑级别，b 代表是否应用皮肤过滤器，a=2，b=1。
void Resize1(char* input,char* output,int w, int h)	图像模糊，w=713，h=467。
void Resize2(char* input,char* output,int w, int	图像模糊。

h)	
void Shift(char* input,char* output,int ch, float v)	Shift 函数，ch=1，v=0.1。
void RGBtoHSV(char* input,char* output)	RGB 转 HSV。
void HSVtoRGB(char* input,char* output)	HSV 转 RGB。
void RGBtoLCH(char* input,char* output)	RGB 转 LCH。
void LCHtoRGB(char* input,char* output)	LCH 转 RGB。
void ImageCutting(char* input,char* output,int leftdownx,int leftdowny,int rightupx,int rightupy)	图像裁剪，input 是输入文件名，output 是输出文件名。leftdownx, leftdowny, rightupx, rightupy 是要裁剪的矩形区域的左下角和右上角的坐标（连续四个整数值，如 50 50 300 300)。支持 24 位 BMP 图像。
void ImageLayerAlgorithm(char* input,char* output)	图像层算法。
void RGBtoGraywithoutLUT(char* input,char* output)	图像无 LUT 的灰度化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void RGBtoGraywithLUT(char* input,char* output)	图像有 LUT 的灰度化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void PiecewiseLinearTransform(char* input,char* output)	分段线性变换，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void PowerConvertion(char* input,char* output,double c,double g)	功率转换，input 是输入文件名，output 是输出文件名。如：c = 1.2，g = 0.5。支持 8 位 BMP 图像。

void LaplacianEnhancement(char* input,char* output,int N,int LaplMask[3][3])	拉普拉斯图像增强，input 是输入文件名，output 是输出文件名。如：N=1。支持 8 位 BMP 图像。
	参考模板：
	int LaplMask[3][3] = {
	0,1,0,
	1, -4, 1,
	0, 1, 0
	};

void Smooth(char* input,char* output)	平滑，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void LaplaceSmooth(char*	拉普拉斯平滑，input 是输入文件名，

input,char* output,int N,int LaplMask[3][3])	output 是输出文件名。如：N=1。支持 8 位 BMP 图像。
	参考模板：
	int LaplMask[3][3] = {
	0,1,0,
	1, -4, 1,
	0, 1, 0
	};

void Sobel1(char* input,char* output,int N,int SblMask1[3][3],int SblMask2[3][3])	Sobel 算子，input 是输入文件名，output 是输出文件名。如：N=1。支持 8 位 BMP 图像。
	参考模板：
	int SblMask1[3][3] = {
	-1,-2,-1,
	0, 0, 0,
	1, 2, 1
	};
	int SblMask2[3][3] = {
	-1, 0, 1,
	-2, 0, 2,
	-1, 0, 1
	};

void SobelSmooth(char* input,char* output,int N,int SblMask1[3][3],int SblMask2[3][3])	Sobel 平滑，input 是输入文件名，output 是输出文件名。如：N=1。支持 8 位 BMP 图像。
	参考模板：
	int SblMask1[3][3] = {
	-1,-2,-1,
	0, 0, 0,
	1, 2, 1
	};
	int SblMask2[3][3] = {
	-1, 0, 1,
	-2, 0, 2,
	-1, 0, 1
	};

void Multiply(char* input,char* output,int N,int SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3])	图像倍增化，input 是输入文件名，output 是输出文件名。如：N=1。支持 8 位 BMP 图像。 参考模板： int LaplMask[3][3] = { 0,1,0, 1, -4, 1, 0, 1, 0

	};
	int SblMask1[3][3] = {
	-1,-2,-1,
	0, 0, 0,
	1, 2, 1
	};
	int SblMask2[3][3] = {
	-1, 0, 1,
	-2, 0, 2,
	-1, 0, 1
	};

void Add(char* input,char* output,int N,int SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3])	图像添加，input 是输入文件名，output 是输出文件名。如：N=1。支持 8 位 BMP 图像。参考模板：
	int LaplMask[3][3] = {
	0,1,0,
	1, -4, 1,
	0, 1, 0
	};
	int SblMask1[3][3] = {
	-1,-2,-1,
	0, 0, 0,
	1, 2, 1
	};
	int SblMask2[3][3] = {
	-1, 0, 1,
	-2, 0, 2,
	-1, 0, 1
	};

void PowerConvertion1(char* input,char* output,double c,double g,int N,int SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3])	功率变换，input 是输入文件名，output 是输出文件名。如：c = 1.2，g = 0.5，N=1。支持 8 位 BMP 图像。
	int LaplMask[3][3] = {
	0,1,0,
	1, -4, 1,
	0, 1, 0
	};
	int SblMask1[3][3] = {
	-1,-2,-1,
	0, 0, 0,
	1, 2, 1
	};
	int SblMask2[3][3] = {
	-1, 0, 1,

	-2, 0, 2, -1, 0, 1 };
void BlackWhite(char* input,char* output)	黑白化图像，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void RandomOperation(char* input,char* output,unsigned char treshold1,unsigned char treshold2,unsigned char treshold3,unsigned char treshold4,unsigned char treshold5,unsigned char treshold6,unsigned char red,unsigned char green,unsigned char blue,int color1,int color2,int color3,int color4,int color5,int color6,int color7,int color8)	随意操作，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void SpecialEffects1(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue)	图像特效，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void NostalgicFilter(BMPMat** input,BMPMat** output)	怀旧滤镜，支持 24 位 BMP 图像。
void SizeTransformation(short** input,short** output,short height,short width,short out_height,short out_width)	图像放缩，支持 8 位 BMP 图像。
void ReverseColor(short** input,short** output,long height,long width,short GRAY_LEVELS)	图像反色。
void Logarithm(short** input,short** output,long height,long width,short c)	对数变换，默认 c=10。
void Gamma(short** input,short** output,long height,long width,double c)	幂律（伽马）变换，默认 c=1.2。
void HistogramEqualization(short** input, short** output, long	直方图均衡化。

height, long width,short GRAY_LEVELS)	
void SmoothLinearFiltering(short** input, short** output,long height, long width,short average[3][3])	平滑线性滤波器。
void MedianFiltering(short** input, short** output, long height, long width)	中值滤波器。
void Laplace(short** input,short** output,long height,long width,short sharpen[3][3])	拉普拉斯算子。
void Sobel(short** input,short** output,long height,long width,short soble1[3][3],short soble2[3][3])	Sobel 算子。
void DFTRead(short** input, double** output,long height,long width)	二维离散傅里叶变换，实部图像。
void DFTImaginary(short** input,double** output,long height,long width)	二维离散傅里叶变换，虚部图像。
void FreSpectrum(short **input, short **output,long height,long width)	傅里叶变换的平移。
void IDFT(double** re_array,double** im_array,short** output,long height,long width)	二维离散傅里叶反变换。
void AddGaussianNoise(short** input,short** output,long height,long width)	添加高斯噪声。
void AddSaltPepperNoise(short** input, short** output,long height,long width)	添加椒盐噪声。
void MeanFilter(short** input,short** output,long height,long width)	均值滤波器。
void	几何均值滤波器，默认 product=1.0。

GeometricMeanFilter(short** input,short** output,long height,long width,double product)	
void HarmonicMeanFiltering(short** input,short** output,long height,long width,double sum)	谐波均值滤波，默认 sum=0。
void InverseHarmonicMeanFiltering(short** input,short** output,long height,long width,int Q)	逆谐波均值滤波，Q 为滤波器的阶数，Q 为正时，消除胡椒噪声，Q 为负时消除盐粒噪声，Q=0 为算术均值滤波器，Q=-1 谐波均值滤波器，默认 Q=2。
void Threshold(short** input,short** output,long height,long width,int delt_t,double T)	基本全局阈值处理方法。
void OTSU(short** input,short** output,long height,long width,short GRAY_LEVELS)	Otsu 方法进行最佳全局阈值处理。
void MatrixGlobalAddition24(BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相加。
void MatrixGlobalSubtraction24(BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相减。
void MatrixGlobalMultiplication24(BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相乘。
void MatrixGlobalDivision24(BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相除。
void MatrixGlobalAddition32(BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相加。
void MatrixGlobalSubtraction32(BM	基于模板矩阵的全局相减。

PMat** input1,BMPMat** input2,BMPMat** output)	
void MatrixGlobalMultiplication32 (BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相乘。
void MatrixGlobalDivision32(BMPMat** input1,BMPMat** input2,BMPMat** output)	基于模板矩阵的全局相除。
void MatrixGlobalAddition8(unsigned char** input1,unsigned char** input2,unsigned char** output)	基于模板矩阵的全局相加。
void MatrixGlobalSubtraction8(unsigned char** input1,unsigned char** input2,unsigned char** output)	基于模板矩阵的全局相减。
void MatrixGlobalMultiplication8(unsigned char** input1,unsigned char** input2,unsigned char** output)	基于模板矩阵的全局相乘。
void MatrixGlobalDivision8(unsigned char** input1,unsigned char** input2,unsigned char** output)	基于模板矩阵的全局相除。
void ColorRectangleLocalSegmentation(char* input,char* output,int x1,int y1,int x2,int y2,BMPMat color)	彩色图以矩形方式局部截取且其它部分填充，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。 函数源代码： 需引入以下头文件： typedef struct { unsigned char B; unsigned char G; unsigned char R; unsigned char A; }BMPMat; 声明： unsigned char** BMPRead8(char*

	input);
	void GenerateImage8(char* output,unsigned char** color);
	BMPMat** BMPRead(char* input);
	void GenerateImage(char* output,BMPMat** color,unsigned short type);
	unsigned int BMPHeight(char* input);
	unsigned int BMPWidth(char* input);
	参考例程：
	BMPMat color={255,255,255};
	BMPMat** input=BMPRead(inputfile);
	BMPMat** output=BMPRead(inputfile);
	unsigned int height=BMPHeight(inputfile);
	unsigned int width=BMPWidth(inputfile);
	for(unsigned int i = 0;i<height;i++){
	for(unsigned int j = 0;j<width;j++){
	output[i][j].B=color.B;
	output[i][j].G=color.G;
	output[i][j].R=color.R;
	}
	}
	for(unsigned int i = y1;i<=y2;i++){
	for(unsigned int j = x1;j<=x2;j++){
	output[i][j].B=input[i][j].B;
	output[i][j].G=input[i][j].G;
	output[i][j].R=input[i][j].R;
	}
	}
	GenerateImage(outputfile,output,24);

void GrayRectangleLocalSegmentati	灰度图以矩形方式局部截取且其它部分填充，（x1，y1）是矩形的左上角的坐标，

on(char* input,char* output,int x1,int y1,int x2,int y2,unsigned char color)	（x2,y2）是矩形的右下角的坐标。
	函数源代码：
	需引入以下头文件：
	typedef struct {
	unsigned char B;
	unsigned char G;
	unsigned char R;
	unsigned char A;
	}BMPMat;
	声明：
	unsigned char** BMPRead8(char* input);
	void GenerateImage8(char* output,unsigned char** color);
	BMPMat** BMPRead(char* input);
	void GenerateImage(char* output,BMPMat** color,unsigned short type);
	unsigned int BMPHeight(char* input);
	unsigned int BMPWidth(char* input);
	参考例程：
	unsigned char color=255;
	unsigned char** input=BMPRead8(inputfile);
	unsigned char** output=BMPRead8(inputfile);
	unsigned int height=BMPHeight(inputfile);
	unsigned int width=BMPWidth(inputfile);
	for(unsigned int i=0;i<height;i++){
	for(unsigned int j=0;j<width;j++){
	output[i][j]=color;
	}
	}
	for(unsigned int i=y1;i<=y2;i++){
	for(unsigned int j=x1;j<=x2;j++){
	output[i][j]=input[i][j];
	}
	}

	GenerateImage8(outputfile,output);

void ColorDrawRectangle(char* input,char* output,int x1,int y1,int x2,int y2,BMPMat color)	彩色图画矩形，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。
	函数源代码：
	需引入以下头文件：
	typedef struct {
	unsigned char B;
	unsigned char G;
	unsigned char R;
	unsigned char A;
	}BMPMat;
	声明：
	unsigned char** BMPRead8(char* input);
	void GenerateImage8(char* output,unsigned char** color);
	BMPMat** BMPRead(char* input);
	void GenerateImage(char* output,BMPMat** color,unsigned short type);
	unsigned int BMPHeight(char* input);
	unsigned int BMPWidth(char* input);
	参考例程：
	BMPMat color={255,255,255};
	BMPMat** input=BMPRead(inputfile);
	BMPMat** output=BMPRead(inputfile);
	unsigned int height=BMPHeight(inputfile);
	unsigned int width=BMPWidth(inputfile);
	for(unsigned int i=0;i<height;i++){
	for(unsigned int j=0;j<width;j++){
	output[i][j].B=color.B;
	output[i][j].G=color.G;
	output[i][j].R=color.R;
	}
	}
	for(unsigned int i=0;i<height;i++){

	for(unsigned int j = 0;j<width;j++){
	if(j>=x1&&j<=x2&&i==y1)
	{
	output[i][j].B=color.B;
	output[i][j].G=color.G;
	output[i][j].R=color.R;
	}
	if(j==x1&&i>=y1&&i<=y2)
	{
	output[i][j].B=color.B;
	output[i][j].G=color.G;
	output[i][j].R=color.R;
	}
	if(j==x2&&i>=y1&&i<=y2)
	{
	output[i][j].B=color.B;
	output[i][j].G=color.G;
	output[i][j].R=color.R;
	}
	if(j>=x1&&j<=x2&&i==y2)
	{
	output[i][j].B=color.B;
	output[i][j].G=color.G;
	output[i][j].R=color.R;
	}
	}
	}
	
	GenerateImage(outputfile,output,24);

void GrayDrawRectangle(char* input,char* output,int x1,int y1,int x2,int y2,unsigned char color)	灰度图画矩形，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。
	函数源代码：
	需引入以下头文件：
	typedef struct {
	unsigned char B;
	unsigned char G;
	unsigned char R;
	unsigned char A;
	}BMPMat;
	声明：
	unsigned char** BMPRead8(char* input);
	void GenerateImage8(char*

output,unsigned char** color);
BMPMat** BMPRead(char* input);
void GenerateImage(char* output,BMPMat** color,unsigned short type);
unsigned int BMPHeight(char* input);
unsigned int BMPWidth(char* input);
参考例程：

```
    unsigned char color=255;
    unsigned char** input=BMPRead8(inputfile);
    unsigned char** output=BMPRead8(inputfile);
    unsigned int height=BMPHeight(inputfile);
    unsigned int width=BMPWidth(inputfile);
    for(unsigned int i = 0;i<height;i++){
        for(unsigned int j = 0;j<width;j++){
           output[i][j]=color;
        }
    }
    for(unsigned int i = 0;i<height;i++){
        for(unsigned int j = 0;j<width;j++){
          if(j>=x1&&j<=x2&&i==y1)
          {
           output[i][j]=color;
           }
           if(j==x1&&i>=y1&&i<=y2)
          {
           output[i][j]=color;
           }
           if(j==x2&&i>=y1&&i<=y2)
          {
           output[i][j]=color;
           }
           if(j>=x1&&j<=x2&&i==y2)
          {
           output[i][j]=color;
           }
```

	} } GenerateImage8(outputfile,output);
void Relief(BMPMat** input,BMPMat** output,int value)	浮雕效果，默认 value=128。
void Relief(unsigned char** input,unsigned char** output,int value)	浮雕效果，默认 value=128。
void Sharpening(BMPMat** input,BMPMat** output,double degree)	图像锐化，默认 degree=0.3。
void Sharpening(unsigned char** input,unsigned char** output,double degree)	图像锐化，默认 degree=0.3。
void Soften(BMPMat** input,BMPMat** output,int value)	图像柔化，默认 value=9。
void Soften(unsigned char** input,unsigned char** output,int value)	图像柔化，默认 value=9。
void flipX(char* input,char* output)	X 方向翻转，支持 JPG 文件。
void flipY(char* input,char* output)	Y 方向翻转，支持 JPG 文件。
void Crop(char* input,char* output,uint16_t start_x, uint16_t start_y, uint16_t new_height, uint16_t new_width)	裁剪。
void Resize(char* input,char* output,int new_width, int new_height)	缩放。
void Scale(char* input,char* output,double ratio)	比例。
void GrayscaleAvg(char* input,char* output)	灰度平均值。
void grayscaleLum(char* input,char* output)	灰度亮度。
void ColorMask(char* input,char* output,float r,float g,float b)	彩色遮罩。
void PixeLize(char*	像素化，参考：strength=2。

input,char* output,int strength)	
void GaussianBlur(char* input,char* output,int strength)	高斯模糊，参考：strength=2。
void EdgeDetection(char* input,char* output,double cutoff)	边缘检测，参考：cutoff=115。
void Sharpen(char* input,char* output)	锐化。
void GrayAVS(char* input,char* output,float k,float b)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void HistogramEqualize24(char* input,char* output)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void MatrixTransformation(char* input,char* output)	矩阵变换。
void Binarization(char* input,char* output)	二值化。
void ChannelSeparation_B(char* input,char* output)	分离出蓝色通道。
void ChannelSeparation_G(char* input,char* output)	分离出绿色通道。
void ChannelSeparation_R(char* input,char* output)	分离出红色通道。
void Inverse(char* input,char* output)	反转。
void HistogramEqualization8(char* input,char* output)	直方图均衡化。
void Smooth(char* input,char* output)	平滑。
void CannyEdge(char* input,char* output)	Canny 算子。
void EdgeEnhance(char* input,char* output)	边缘增强。
void AvrFilter(char* input,char* output1,char* output2,int M,int N)	input 是输入文件名，output 是输出文件名。如 M=21，N=1。支持 8 位 BMP 图像。

void GryOppositionSSE(char* input,char* output)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void MedianFilter(char* input,char* output,int M,int N)	中值滤波器，input 是输入文件名，output 是输出文件名。如 M=5，N=5。支持 8 位 BMP 图像。
void EdgeSharpeningGry(char* input,char* output)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void SJGryandRiceTest(char* input,char* output)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void TextTest(char* input,char* output)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void RedChannel(char* input,char* output)	生成图像的红色通道图像，input 是输入文件名,output 是输出文件名。支持 24 位 BMP 图像。
void GreenChannel(char* input,char* output)	生成图像的绿色通道图像，input 是输入文件名,output 是输出文件名。支持 24 位 BMP 图像。
void BlueChannel(char* input,char* output)	生成图像的蓝色通道图像，input 是输入文件名,output 是输出文件名。支持 24 位 BMP 图像。
void HistogramStatistics(char* input,char* output)	直方图统计，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void HistogramEqualization1(char* input,char* output)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void ReflectionRay(char* input,char* output)	反射线，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void MeanFiltering24(char* input,char* output)	均值滤波，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void MedianFiltering24(char* input,char* output)	中值滤波，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void ZoomOutAndZoomIn(char* input,char* output,double value)	缩放（双线性插值），input 是输入文件名，output 是输出文件名。value 是放大倍数，如 value=0.5。支持 24 位 BMP 图像。
void Translation24(char* input,char* output,int x,int y)	平移，input 是输入文件名，output 是输出文件名。x 是横轴的平移量，y 是纵轴的平移量，如 x=-10，y=-30。支持 24 位 BMP 图像。
void Mirror24(char* input,char* output)	镜像，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void Rotate24(char* input,char* output,double degree)	旋转，input 是输入文件名，output 是输出文件名。degree 是旋转的度数。支持 24 位 BMP 图像。

void GivenThresholdMethod(char* input,char* output,int threshold)	给定阈值法处理图像,使图片黑白化,input 是输入文件名，output 是输出文件名。threshold 是给定的阈值，如 threshold=100。支持 24 位 BMP 图像。
void IterativeThresholdMethod(char* input,char* output)	迭代阈值法处理图像,使图片黑白化,input 是输入文件名,output 是输出文件名。支持 24 位 BMP 图像。
void OstuThresholdSegmentationMethod(char* input,char* output)	Ostu（大津法）阈值分割，input 是输入文件名,output 是输出文件名。支持 24 位 BMP 图像。
void Repudiation(char* input,char* output)	将伪彩图片反白，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void Gray1(char* input,char* output)	将彩色图片变成灰度图片，input 是输入文件名,output 是输出文件名。支持 24 位 BMP 图像。
void CorrectMethod(char* input,char* output)	正确法，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void ChannelSeparation1(char* input,char* Routput,char* Goutput,char* Boutput)	对图像分理出其中的 RGB 分量并分别保存为独立的图像，input 是输入文件名，Routput 是红色通道图像，Goutput 是绿色通道图像，Boutput 是绿色通道图像。支持 24 位 BMP 图像。
void ReverseColor(char* input,char* output)	对灰度图进行反色，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。

Image1* LoadImage1(char* input)	BMP 图像读取，input 是输入文件名。支持 8 位和 24 位 BMP 图像。
	返回 Image1 型数据，Image1 型数据的结构如下：
	typedef struct
	{
	int width;
	int height;
	int channels; //图像通道数
	unsigned char* Data; //像素数据
	}Image1;

void SaveImage1(char* output,Image1* img)	将 Image1 型数据保存为 BMP 图像，output 是生成的 BMP 图像文件名，img 是要保存的图像数据。支持 8 位和 24 位 BMP 图像。 Image1 型数据的结构如下： typedef struct { int width;

	int height;
	int channels; //图像通道数
	unsigned char* Data; //像素数据
	}Image1;

void ImageContrastExtension(char* input,char* output,double m,double g1,double g2,double a)	图像对比度扩展，input 是输入文件名，output 是输出文件名。 其中，可参考：double m=1.5,g1=100.0,g2=200.0; m 对应斜率 double a=(255.0-m*(g2-g1))/(255.0-(g2-g1)); 支持 8 位 BMP 图像。
void Binaryzation(char* input,char* output,int threshold)	图像二值化，input 是输入文件名，output 是输出文件名。threshold 是将灰度值转化为二值的阈值，如 threshold=80。支持 24 位 BMP 图像。
void GlobalBinarization(char* input,char* output)	全局二值化，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void AdaptiveBinarization(char* input,char* output)	自适应二值化，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void ExpansionOperation(char* input,char* output)	膨胀操作，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void CorrosionOperation(char* input,char* output)	腐蚀操作，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Operation1(char* input,char* output)	开操作，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Closed1(char* input,char* output)	闭操作，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Negative1(char* input,char* output)	图像反色，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
void Negative(char* input,char* output)	图像反色，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void ImageSynthesis(char* input1,char* input2,char* output)	图像合成。
void BlackWhite(char* input,char* output,float T,int border)	黑白化，支持 8 位和 24 位 BMP 图像。T 是阈值，border 是边界范围，如：T=50，border=0。
IMAGE Image_bmp_load(char* filename)	加载 BMP 图片。
void Image_bmp_save(char*	保存 BMP 图片。

filename,IMAGE im)	
IMAGE TransformShapeNearest(IMAGE input, unsigned int newWidth, unsigned int newHeight)	缩放图片(最近邻插值法)。
IMAGE TransformShapeLinear(IMAGE input, unsigned int newWidth, unsigned int newHeight)	缩放图片(双线性插值法)。
IMAGE TransformShapeWhirl(IMAGE input, float angle)	图像的任意角度的旋转。
IMAGE TransformShapeUpturn(IMAGE input, int a)	图像的镜像翻转。
void TransformColorGrayscale(IMAGE im, int GrayscaleMode)	彩色图转灰度图，对于 GrayscaleMode 的值：1 表示加权法，2 表示最值法，3 表示均值法，4 表示红色分量法，5 表示绿色分量法，6 表示蓝色分量法。
void TransformColorBWDIY(IMAGE input, unsigned char Threshold)	二值图(自定义阈值法)。
void TransformColorBWOSTU(IMAGE input)	二值图(大津法 OSTU，适用双峰直方图。)
void TransformColorBWTRIANGLE(IMAGE input)	二值图（三角法 TRIANGLE，适用单峰直方图。）
IMAGE TransformColorBWAdaptive(IMAGE input, int areaSize)	二值图(自适应阈值法，areaSize=25 较合适)
IMAGE TransformColorBWGrayscale(IMAGE input, int areaSize)	二值图(用二值图表示灰度变化,areaSize=25 较合适)
void TransformColorOpposite(IMAGE input)	反色。
IMAGE TransformColorHistogramPart(IMAGE input)	直方图均衡化（分步计算，效果更加柔和）。
IMAGE TransformColorHistogramAll(IMAGE input)	直方图均衡化（整体计算，效果更加尖锐）。

IMAGE KernelsUseDIY(IMAGE input, double* kernels, int areaSize, double modulus)	卷积操作（自定义）。
IMAGE WavefilteringMedian(IMAGE input)	中值滤波。

IMAGE WavefilteringGauss(IMAGE input,double KERNELS_Wave_Gauss[9],int a,double b)	高斯滤波。
	高斯滤波卷积核：
	double KERNELS_Wave_Gauss[9] =
	{
	1, 2, 1,
	2, 4, 2,
	1, 2 ,1
	};

IMAGE Wavefiltering_LowPass(IMAGE input, double* kernels)	低通滤波。
	// 低通滤波卷积核 LP1
	double KERNELS_Wave_LowPass_LP1[9] =
	{
	1 / 9.0, 1 / 9.0, 1 / 9.0,
	1 / 9.0, 1 / 9.0, 1 / 9.0,
	1 / 9.0, 1 / 9.0, 1 / 9.0
	};
	
	// 低通滤波卷积核 LP2
	double KERNELS_Wave_LowPass_LP2[9] =
	{
	1 / 10.0, 1 / 10.0, 1 / 10.0,
	1 / 10.0, 1 / 5.0, 1 / 10.0,
	1 / 10.0, 1 / 10.0, 1 / 10.0
	};
	
	// 低通滤波卷积核 LP3
	double KERNELS_Wave_LowPass_LP3[9] =
	{
	1 / 16.0, 1 / 8.0, 1 / 16.0,
	1 / 8.0, 1 / 4.0, 1 / 8.0,
	1 / 16.0, 1 / 8.0, 1 / 16.0
	};

IMAGE WavefilteringHighPass(IMAGE input, double* kernels)	高通滤波。
	// 高通滤波卷积核 HP1
	double KERNELS_Wave_HighPass_HP1[9] =
	{
	-1, -1, -1,
	-1, 9, -1,

	-1, -1 ,-1
	};
	
	// 高通滤波卷积核 HP2
	double KERNELS_Wave_HighPass_HP2[9] =
	{
	0, -1, 0,
	-1, 5, -1,
	0, -1 ,0
	};
	
	// 高通滤波卷积核 HP3
	double KERNELS_Wave_HighPass_HP3[9] =
	{
	1, -2, 1,
	-2, 5, -2,
	1, -2 ,1
	};

IMAGE Wavefiltering_Average(IMAGE input,double* KERNELS_Wave_Average)	均值滤波。
	// 均值滤波卷积核
	double KERNELS_Wave_Average[25] =
	{
	1, 1, 1, 1, 1,
	1, 1, 1, 1, 1,
	1, 1, 1, 1, 1,
	1, 1, 1, 1, 1,
	1, 1, 1, 1, 1
	};

IMAGE EdgeDetectionDifference(IMAGE input, double* kernels)	差分边缘检测。
	// 差分垂直边缘检测卷积核
	double KERNELS_Edge_difference_vertical[9] =
	{
	0, 0, 0,
	-1, 1, 0,
	0, 0, 0
	};
	
	// 差分水平边缘检测卷积核
	double KERNELS_Edge_difference_horizontal[9] =
	{
	0,-1, 0,

	0, 1, 0,
	0, 0, 0
	};
	
	// 差分垂直和水平边缘检测卷积核
	double KERNELS_Edge_difference_VH[9]
	=
	{
	-1, 0, 0,
	0, 1, 0,
	0, 0, 0
	};

IMAGE KernelsUseEdgeSobel(IMAGE input, double* kernels1, double* kernels2)	Sobel 边缘检测。
	// Sobel X 边缘检测卷积核
	double KERNELS_Edge_Sobel_X[9] =
	{
	-1, 0, 1,
	- 2, 0, 2,
	-1, 0, 1
	};
	
	// Sobel Y 边缘检测卷积核
	double KERNELS_Edge_Sobel_Y[9] =
	{
	-1, -2, -1,
	0, 0, 0,
	1, 2, 1
	};

IMAGE EdgeDetectionLaplace(IMAGE input, double* kernels)	Laplace 边缘检测。
	// Laplace 边缘检测卷积核 LAP1
	double KERNELS_Edge_Laplace_LAP1[9] =
	{
	0, 1, 0,
	1, -4, 1,
	0, 1, 0
	};
	
	// Laplace 边缘检测卷积核 LAP2
	double KERNELS_Edge_Laplace_LAP2[9] =
	{
	-1, -1, -1,
	-1, 8, -1,
	-1, -1, -1
	};

	// Laplace 边缘检测卷积核 LAP3
	double KERNELS_Edge_Laplace_LAP3[9] =
	{
	-1, -1, -1,
	-1, 9, -1,
	-1, -1, -1
	};
	
	// Laplace 边缘检测卷积核 LAP4
	double KERNELS_Edge_Laplace_LAP4[9] =
	{
	1, -2, 1,
	-2, 8, -2,
	1, -2, 1
	};

IMAGE MorphologyErosion(IMAGE input, double* kernels)	腐蚀。
	// 腐蚀卷积核
	double KERNELS_Morphology_Erosion_cross[9] =
	{
	0, 1, 0,
	1, 1, 1,
	0, 1, 0
	};

IMAGE MorphologyDilation(IMAGE input, double* kernels)	膨胀。
	// 膨胀卷积核
	double KERNELS_Morphology_Dilation_cross[9] =
	{
	0, 1, 0,
	1, 1, 1,
	0, 1, 0
	};

IMAGE Pooling(IMAGE input, int lenght)	池化。
IGIMAGE IntegralImage(IMAGE input)	获得积分图（在此之前要保证图片是“白底黑字”）。
void FaceDetection(char* input,char* output,double* KERNELS_Wave_Average)	人脸检测。
IMAGE FaceDetection(IMAGE input1,IMAGE input2,double*	人脸检测。 需引入以下结构体：

KERNELS_Wave_Average)	typedef struct tagBGRA
	{
	unsigned char blue;
	unsigned char green;
	unsigned char red;
	unsigned char transparency;
	}BGRA, *PBGRA;
	typedef struct tagIMAGE
	{
	unsigned int w;
	unsigned int h;
	BGRA* color;
	}IMAGE, *PIMAGE;
	声明：
	IMAGE Image_bmp_load(char* filename);
	void Image_bmp_save(char* filename,IMAGE im);
	参考：
	// 用于处理
	IMAGE input2 = Image_bmp_load(inputfile);
	// 用于保存
	IMAGE input2= Image_bmp_load(inputfile);
	input2=FaceDetection(input1,input2,KERNELS_Wave_Average);
	// 保存图片
	Image_bmp_save(outputfile, input2);

void IntegralDiagram(unsigned int *input, unsigned int *output, int width, int height)	图像积分图。
void ImageEncryption(char* inFileName,char* outFileName,char key)	图像加密，支持 8 位、24 位和 32 位 BMP 图像。inFileName 是原图图像文件名，outFileName 是解密图像文件名，key 是密钥，如 key=255。
void ImageDecryption(char* inFileName,char* outFileName,char key)	图像解密，inFileName 是加密图像文件名，outFileName 是解密图像文件名，key 是密钥，如 key=255。支持 8 位、24 位和 32 位 BMP 图像。
void EncryptionDecryption(char* input,char* output,int Key,int a)	图像加解密，Key 是密钥，a=1 时执行加密，a=0 时执行解密。支持 24 位 BMP 图像。

void Encryption(char* input,char* output,int Key)	图像加密，input 是输入文件名，output 是输出文件名。Key 是密钥。支持 24 位 BMP 图像。
void Decryption(char* input,char* output,int Key)	图像解密，input 是输入文件名，output 是输出文件名。Key 是密钥。支持 24 位 BMP 图像。
void Compress8(string input,string output)	图像压缩，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void Decompression(string input,string output)	图像解压，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像压缩后的结果文件。
void HorizontalMirror(char* input,char* output)	水平镜像，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void MirrorVertically(char* input,char* output)	垂直镜像，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void XMirroring(char* input,char* output)	X 镜像，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void YMirroring(char* input,char* output)	Y 镜像，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void ImageConvolution(char* input,char* output,double** Kernel,int n,int m)	图像卷积，input 是输入文件名，output 是输出文件名。Kernel 是卷积核，如 double Kernel[3][3] = {{-0.225,-0.225-0.225},{-0.225,1,-0.225},{-0.225,-0.225,-0.225}}；n 是 Kernel 的第一维的大小，m 是 Kernel 的第二维的大小，形如 Kernel[n][m]。支持 24 位 BMP 图像。
void SpatialMeanFiter(char* input,char* output,int radius)	空间均值过滤器，参考：radius=3。
void SpatialMedianFiter(char* input,char* output,int radius)	空间中值过滤器，参考：radius=3。
void SpatialMaxFiter(char* input,char* output,int radius)	空间最大过滤器，参考：radius=3。
void SpatialMinFiter(char* input,char* output,int radius)	空间最小过滤器，参考：radius=3。
void SpatialGaussFiter(char* input,char* output,int radius)	空间高斯过滤器，参考：radius=3。
void SpatialStatisticalFiter(char	空间统计滤波器，参考：radius=3，T=0.2。

* input,char* output,int radius,float T)	
void Mosaic(char* input,char* output,int w,int h)	马赛克滤镜，w 和 h 是输出图像的宽和高。支持 PNG 图像。
void FFTAmp(char* input,char* output,bool inv)	FFT 放大器，参考：inv=false。
void FFTPhase(char* input,char* output,bool inv)	FFT 相位，参考：inv=false。
void STDFT1(char* input,char* output,bool inv)	参考：inv=false。
void STDFT2(char* input,char* output,bool inv)	参考：inv=false。
void SpectrumShaping(char* input,char* inputMsk,char* output)	图像频域滤波，FFT 变换--相位谱，inputMsk 是输入的掩膜图像名。
void Translation(char* input,char* output,int x,int y,unsigned char color)	图像平移，input 是输入文件名，output 是输出文件名。x 和 y 是在 X 轴和 Y 轴平移的量，以右为正向，color 是平移后非原图区域填充的颜色,如 color=100。支持 8 位 BMP 图像。
void Nesting(char* Biginput,char* Smallinput,char* output)	图像嵌套，Biginput 是输入的大图，Smallinput 是输入的小图。支持 24 位 BMP 图像。
void CrossDenoising24(BMPMat** input,BMPMat** output,BMPMat threshold,BMPMat target)	图像去除某种像素，output 用于保存结果（与 input 大小相同）。
void CrossDenoising8(unsigned char** input,unsigned char** output,unsigned char threshold,unsigned char target)	图像去除某种像素，output 用于保存结果（与 input 大小相同）。
void ImageDecontamination(BMPMat** input,BMPMat** output,int x1,int y1,int x2,int y2)	图像去污。(x1,y1)是矩形污渍区的左上角坐标,(x2,y2)是矩形污渍区的右下角坐标。
void ImageDecontamination(unsigned char** input,unsigned char** output,int x1,int y1,int x2,int y2)	图像去污。(x1,y1)是矩形污渍区的左上角坐标,(x2,y2)是矩形污渍区的右下角坐标。
void Blend(char*	图像融合之混合化，input1 和 input2 是输

input1,char* input2,char* output)	入的两个要融合的图像，output 是输出文件名。支持 24 位 BMP 图像。
void Checker(char* input1,char* input2,char* output)	图像融合之棋盘化，input1 和 input2 是输入的两个要融合的图像，output 是输出文件名。支持 24 位 BMP 图像。
void Blend1(char* input1,char* input2,char* output)	图像融合之混合化，input1 和 input2 是输入的两个要融合的图像，output 是输出文件名。支持 24 位 BMP 图像。
void Checker1(char* input1,char* input2,char* output)	图像融合之棋盘化，input1 和 input2 是输入的两个要融合的图像，output 是输出文件名。支持 24 位 BMP 图像。
void ImageSharpening(char* input,char* output)	图像锐化，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void SharpenLaplace(char* input,char* output,int ratio)	拉普拉斯锐化，参考：ratio=100。
void SharpenUSM(char* input,char* output,int radius,int amount,int threshold)	USM 锐化，参考：radius=5，amount=400，threshold=50。
void DrawRectangle(char* input,char* output,int x1,int y1,int x2,int y2,unsigned char red,unsigned char green,unsigned char blue)	在 24 位 BMP 图像上通过传入的参数画一个矩形。input 是输入文件名，output 是输出文件名。(x1,y1)是矩形坐上顶点的坐标，(x2,y2)是矩形右下顶点的坐标；red 是矩形线框的红色分量，green 是矩形线框的绿色分量，blue 是矩形的蓝色分量。
void GenerateBmp(unsigned char* pData,int width,int height,char* filename)	生成 BMP 图像，pData 是图像的像素数据，width 和 height 是图像的宽和高,filename 是生成的图像的文件名。
void Jpg24ImageGeneration(char* filename,unsigned int width, unsigned int height, unsigned char* img)	JPG 图像生成,filename 是生成的 JPG 图像文件名，width 是图像的宽，height 是图像的高，img 是图像的像素数据。
void ImageScalingNearestNeighborInterpolation(char* input,char* output,float lx,float ly)	最近邻插值法去栅格，input 是输入文件名，output 是输出文件名。lx 和 ly 是长和宽需要缩放的倍数。支持 8 位 BMP 图像。
void ImageScalingBilinearInterpolation(char* input,char* output,float lx,float ly)	双线性插值法去栅格，input 是输入文件名，output 是输出文件名。lx 和 ly 是长和宽需要缩放的倍数。支持 8 位 BMP 图像。
void	双线性插值，input 是输入文件名，output

BilinearInterpolationScaling (char* input,char* output,float ExpScalValue)	是输出文件名。ExpScalValue 是期望的缩放倍数（允许小数）。支持 BMP 图像。
void NearestNeighborInterpolation Scaling(char* input,char* output,float ExpScalValue)	最近邻插值，input 是输入文件名，output 是输出文件名。ExpScalValue 是期望的缩放倍数（允许小数）。支持 BMP 图像。
void ZoomImg(unsigned char *input,unsigned char *output,int sw,int sh,int channels,int dw,int dh)	二次线性插值图像缩放。
void CrossDenoising24(BMPMat** input,BMPMat** output,BMPMat target,BMPMatdouble weight)	图像修复，output 用于保存结果（与 input 大小相同），target 是污点像素，weight 是修复权重系数。
void CrossDenoising8(unsigned char** input,unsigned char** output,unsigned char target,double weight)	图像修复，output 用于保存结果（与 input 大小相同），target 是污点像素，weight 是修复权重系数。
void RotateRight90Degrees(char* input,char* output)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像，向右旋转 90 度。
void RotateLeft90Degrees(char* input,char* output)	input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像，向左旋转 90 度。
void ImageRotation(char* input,char* output,double angle)	图像旋转，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。angle 是要旋转的角度。
void Rotation8(char* input,char* output,double Angle,int x1,int y1,int x2,int y2,unsigned char color)	图像旋转，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。Angle 是要旋转的角度数；x1、y1、x2、y2 是旋转所围绕的中心点的坐标，color 是旋转后非原图区域的填充颜色。
void Rotation24(char* input,char* output,double Angle,int x1,int y1,int x2,int y2,unsigned char red,unsigned char green,unsigned char blue)	图像旋转，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。Angle 是要旋转的角度数；x1、y1、x2、y2 是旋转所围绕的中心点的坐标；red、green、blue 分别是旋转后非原图区域要填充的颜色的红绿蓝分量。
void Rotation(char* input,char* output,int angle,unsigned char color)	图像旋转，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。angle 是旋转的角度，color 是旋转后非原图区域填充的颜色，如 color=100。

void Rotate(char* input,char* output,int angle)	图像旋转，input 是输入文件名，output 是输出文件名。支持 BMP 图像。angle 是旋转的角度。
void imgRotate90Gray(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	灰度图像旋转 90。
void imgRotate90Color(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	彩色图像旋转 90。
void imgRotate270Gray(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	灰阶图像旋转 270。
void imgRotate270Color(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	彩色图像旋转 270。
void imgRotate180Gray(unsigned char *Img,int w,int h)	灰阶图像旋转 180，结果保存在原输入数组中。
void imgRotate180Color(unsigned char *Img,int w,int h)	彩色图像旋转 180，结果保存在原输入数组中。
void imgRBExchange(unsigned char *Img,int w,int h)	彩色图像 R、B 互换，结果保存在原输入数组中。
void NoiseUniform(char* input,char* output,double a,double b)	均匀分布噪声，参考：a=0，b=0.2。
void NoiseGauss(char* input,char* output,float mean,float delta)	高斯噪声，参考：mean=0，delta=31。
void NoiseRayleigh(char* input,char* output,float a,float b)	瑞利噪声，参考：a=0，b=200。
void NoiseExp(char* input,char* output,float a)	指数噪声，参考：a=0.1。
void NoiseImpulse(char* input,char* output,float	椒盐噪声，参考：a=0.2，b=0.2。

a,float b)	
void grayToColor(FILE* input,FILE* output)	灰色转伪彩色，input 是输入文件，output 是输出文件。支持 8 位和 24 位 BMP 图像。
void ImageThinning(char* input,char* output,char** str,int n,int m1,int a,int b)	图像细化，input 是输入文件名，output 是输出文件名。支持 4 位 BMP 图像。n 是 str 第一维的大小，m1 是第二维的大小，形如 str[n][m1]；a 和 b 是相关的调节参数，可以为 a=3，b=5。 参考模板： char str[6][8] = { { 0, 0, 0, 0, 0, 0, 0, 0, }, { 255, 0, 255, 0, 0, 255, 0, 0 }, { 255, 0, 255, 255, 0, 255, 0, 255 }, { 255, 255, 255, 0, 0, 255, 255, 255 }, { 255, 0, 255, 255, 0, 255, 255, 255 }, { 0, 255, 255, 255, 255, 255, 255, 255 } };
int MinimumValueOfImagePixels(char* filename)	返回图像像素的最小值，filename 是输入的图像文件名。支持 8 位和 24 位 BMP 图像。
int MaximumValueOfImagePixels(char* filename)	返回图像像素的最大值，filename 是输入的图像文件名。支持 8 位和 24 位 BMP 图像。
float AverageValueOfImagePixels(char* filename)	返回图像像素的均值，filename 是输入的图像文件名。支持 8 位和 24 位 BMP 图像。
double StandardDeviationOfImagePixels(char* filename)	返回图像像素的标准差，filename 是输入的图像文件名。支持 8 位和 24 位 BMP 图像。
double EntropyOfImage(char* filename)	返回图像的熵，支持 8 位和 24 位 BMP 图像。
float* CountTheFrequencyOfPixels(char* filename)	filename 是输入的图像文件名。存储每个像素的频率，像素值为 0~255，返回值数组中的元素序号即为像素值，该序号在数组下的值即为这个像素的频率。支持 8 位和 24 位 BMP 图像。
void Rotate(char* input,char* output,int angle,int interpolation)	图像旋转。参考：angle=80，interpolation=0 或 interpolation=1。
void HSV(char* input,char* output,int h,int s,int v)	图像色调饱和度明度调节，参考：h=120，s=60，v=20。
void OilpaintFilter(char* input,char* output,int	油画滤镜，参考：radius=10，smooth=100。

radius,int smooth)	
unsigned int* CircleDetection(char* input)	圆检测，返回圆心的坐标和半径，返回值数组中第一个元素是圆心 X 坐标,第二个元素是圆心 Y 坐标，第三个元素是圆半径。
void HaloFilter(char* input,char* output,int ratio)	晕角滤镜，参考：ratio=100。
void GrayHistogram(char* input,char* output,int hWidth,int hHeight)	灰度直方图，参考：hWidth=256，hHeight=100。
void RedHistogram(char* input,char* output,int hWidth,int hHeight)	红色通道直方图，参考：hWidth=256，hHeight=100。
void GreenHistogram(char* input,char* output,int hWidth,int hHeight)	绿色通道直方图，参考：hWidth=256，hHeight=100。
void BlueHistogram(char* input,char* output,int hWidth,int hHeight)	蓝色通道直方图，参考：hWidth=256，hHeight=100。
void HistogramEqualization2(char* input,char* output,int imgBit)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 8 位和 24 位 BMP 图像。imgBit 是输入图像的位数。
void HistogramEqualization3(char* input,char* output)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 8 位和 24 位 BMP 图像。
void HistogramEqualization4(char* input,char* output)	直方图均衡化，input 是输入文件名，output 是输出文件名。支持 8 位和 24 位 BMP 图像。input 是输入文件名称，out 是输出文件名称。
void HistogramEqualization(char* input,char* output,int hWidth,int hHeight)	直方图均衡化，参考：hWidth=256，hHeight=100。
void GrayHistogramEqualization(char* input,char* output,int hWidth,int hHeight)	灰度直方图，参考：hWidth=256，hHeight=100。
void RedHistogramEqualization(char* input,char* output,int hWidth,int hHeight)	红色通道直方图，参考：hWidth=256，hHeight=100。
void GreenHistogramEqualization(c	绿色通道直方图，参考：hWidth=256，hHeight=100。

har* input,char* output,int hWidth,int hHeight)	
void BlueHistogramEqualization(char* input,char* output,int hWidth,int hHeight)	蓝色通道直方图，参考：hWidth=256，hHeight=100。
void GrayScaleStretch(char* input,char* output,int hWidth,int hHeight)	灰度级拉伸，参考：hWidth=256，hHeight=100。
void GrayHistagramStretch(char* input,char* output,int hWidth,int hHeight)	灰度直方图拉伸，参考：hWidth=256，hHeight=100。
void RedHistagramStretch(char* input,char* output,int hWidth,int hHeight)	红色通道直方图，参考：hWidth=256，hHeight=100。
void GreenHistagramStretch(char* input,char* output,int hWidth,int hHeight)	绿色通道直方图，参考：hWidth=256，hHeight=100。
void BlueHistagramStretch(char* input,char* output,int hWidth,int hHeight)	蓝色通道直方图，参考：hWidth=256，hHeight=100。
void MedianFiltering1(char* input,char* output)	中值滤波，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void MedianFiltering2(char* input,char* output)	中值滤波，input 是输入文件名，output 是输出文件名。支持 8 位和 24 位 BMP 图像。
void ThresholdProcessing(char* input,char* output,int Threshold)	阈值处理，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。Threshold 是阈值相关参数，如 Threshold=0.001。
void OTSUProcessing(char* input,char* output)	大津法处理，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。
void OBJtoTGA(char* input,char* output,int width,int height)	OBJ 转 TGA。
void ToRIM(char* input,char* output)	一般图像转到 RIM 图像，支持 PNG、JPG 和 TGA 图像。
void ToImage(char* input,char* output,int jpg_quality)	RIM 图像转到一般图像，支持 PNG、JPG 和 TGA 图像。jpg_quality=25。
void	将 1 位深度的单色 BMP 图片转成热敏打印

ImprimanteThermique(char* input,char* output,ARRAY3 skip_cmd,unsigned short PRINTER_TYPE_BMP,unsigned char mode,unsigned int FILE_TYPE_AD,unsigned char a,unsigned char b)	机的位图打印输出,支持的热敏打印机的位图打印指令为ESC *指令。 typedef unsigned char ARRAY3[3];参考:output="output.pbin"，skip_cmd={0x1B, 0x4A, 0x00},PRINTER_TYPE_BMP 是打印机位图打印指令码标识，PRINTER_TYPE_BMP=(0x2A1B)，mode 是打印机位图打印模式，mode=33，FILE_TYPE_AD 是图片类型，"AD"表示广告图片，FILE_TYPE_AD=(0x4441)，a=0x80，b=1。
void WhiteBalance(const char* input,const char* output)	白平衡。
void Sobel(char* input,char* output,double magnScale,double threshold)	Sobel 算子，magnScale=0.35，threshold=130。支持 PGM 和 PBM 图像。
void Canny(char* input,char* output,double magnScale,double lowThreshold,double highThreshold)	Canny 算子，magnScale=0.35，lowThreshold=55，highThreshold=120。支持 PGM 和 PBM 图像。
void BlackWhite(char* input,char* output,int threshold,int background)	黑白化，threshold=100，background=0。支持 PGM 和 PBM 图像。
void ConnectedComponents(char* input,char* output,int threshold,int background,int threshold1)	区域连通，threshold=100，background=0，threshold1=100。支持 PGM 和 PBM 图像。
void CleanImage(char* input,char* output)	清洁图像。支持 PGM 和 PBM 图像。
void NoiseImage(char* input,char* output,float probability)	噪声化图像，probability=0.1。支持 PGM 和 PBM 图像。
void HoughTransformCircle1(char* input,char* output,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1,double gamma1)	圆检测。scale1=1，gamma1=1.0，magnScale=0.5，lowThreshold=85，highThreshold=150，scale=0，gamma=1.0，sigma 和 kernelSize 用于平滑的高斯 5x5 内核，sigma=1.0，kernelSize=5。 如果 scale==0，则值保持不变，但如果它们低于 0 或高于 255，则它们分别设置为 0 或 255。 如果 scale！=0，则对值进行缩放，使得最

	小值为零，最大值为 255。 设置 gamma 值允许指数缩放，启用 gamma==1.0。 支持 PGM 和 PBM 图像。
void HoughTransformCircle2(char* input,char* output,int number,int minDist,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1, double gamma1)	圆检测。scale1=1，gamma1=1.0，magnScale=0.5，lowThreshold=85，highThreshold=150，scale=0，gamma=1.0，sigma 和 kernelSize 用于平滑的高斯 5x5 内核，sigma=1.0,kernelSize=5,number=10 表示图像的目视检查有 10 个圆圈，minDist=35。 如果 scale==0，则值保持不变，但如果它们低于 0 或高于 255，则它们分别设置为 0 或 255。 如果 scale！=0，则对值进行缩放，使得最小值为零，最大值为 255。 设置 gamma 值允许指数缩放，启用 gamma==1.0。 支持 PGM 和 PBM 图像。
double** HoughTransformCircle3(char* input,char* output,int number,int minDist,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1, double gamma1)	圆检测。scale1=1，gamma1=1.0，magnScale=0.5，lowThreshold=85，highThreshold=150，scale=0，gamma=1.0，sigma 和 kernelSize 用于平滑的高斯 5x5 内核，sigma=1.0,kernelSize=5,number=10 表示图像的目视检查有 10 个圆圈，minDist=35。 如果 scale==0，则值保持不变，但如果它们低于 0 或高于 255，则它们分别设置为 0 或 255。 如果 scale！=0，则对值进行缩放，使得最小值为零，最大值为 255。 设置 gamma 值允许指数缩放，启用 gamma==1.0。 返回以（vCenter，hCenter）和半径（vRadius，hRadius）为中心的椭圆数据，共有 number 组数据，每组包含一个椭圆数据，第一个元素是 vCenter，第二个元素是 hCenter，第三个元素是 vRadius，第四个元素是 hRadius。 支持 PGM 和 PBM 图像。
void ShapeEdgeDetection1(char* input,char* output,unsigned char CANNY_THRESH4,int	形状边缘检测，CANNY_THRESH4=35，CANNY_blur4=7。支持 PNG 图像。

CANNY_blur4)	
void ShapeEdgeDetection2(char* input,char* output,unsigned char CANNY_THRESH4,int CANNY_blur4)	形状边缘检测，CANNY_THRESH4=35，CANNY_blur4=7。支持 PNG 图像。
void ShapeEdgeDetection3(char* input,char* output,unsigned char CANNY_THRESH,int CANNY_BLUR)	形状边缘检测，CANNY_THRESH=50，CANNY_BLUR=12。支持 PNG 图像。
void ShapeEdgeDetection4(char* input,char* output,unsigned char CANNY_THRESH,int CANNY_BLUR)	形状边缘检测，CANNY_THRESH=50，CANNY_BLUR=12。支持 PNG 图像。
void ShapeEdgeDetection5(char* input,char* output,unsigned char CANNY_THRESH2,int CANNY_BLUR2)	形状边缘检测，CANNY_THRESH2=10，CANNY_BLUR2=2。支持 PNG 图像。
void ShapeEdgeDetection6(char* input,char* output,unsigned char CANNY_THRESH2,int CANNY_BLUR2)	形状边缘检测，CANNY_THRESH2=10，CANNY_BLUR2=2。支持 PNG 图像。
void ShapeEdgeDetection7(char* input,char* output,unsigned char CANNY_THRESH3,int CANNY_blur3)	形状边缘检测，CANNY_THRESH3=45，CANNY_blur3=10。支持 PNG 图像。
void ShapeEdgeDetection8(char* input,char* output,unsigned char CANNY_THRESH3,int CANNY_blur3)	形状边缘检测，CANNY_THRESH3=45，CANNY_blur3=10。支持 PNG 图像。

其他处理

void DES_Encrypt(char *PlainFile, char *Key,char *CipherFile)	DES 加密函数，支持多种文件。PlainFile 是原文件的文件名,Key 是密钥字符,CipherFile 是加密后的文件名。
void DES_Decrypt(char *CipherFile, char *Key,char *PlainFile)	DES 解密函数，支持多种文件。CipherFile 是已加密文件的文件名，

	Key 是密钥字符,PlainFile 是解密后的文件名。
void Encode(char* input,char* output)	文本文件压缩，input 是输入文件名，output 是输出文件名。
void Decode(char* input,char* output)	文本文件压缩结果解压缩，input 是输入文件名，output 是输出文件名。
void FileCompress(char *input , char *output)	文件压缩，input 是输入文件名，output 是输出文件名。
void FileDecompression(char *input , char *output)	文件压缩结果解压缩，input 是输入文件名，output 是输出文件名。

高级算子

int* TemplateMatch(char* input,char* Template,char* output,int channels,int ROTATION)	模板匹配，input 是母本图像，Template 是样本图像，output 是结果图像文件名。channels 是图像的像素通道数,返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的 X 轴坐标值，第三个元素是目标的 Y 轴坐标值。至少支持 PNG 图像。参考：channels=3，ROTATION=360 或 ROTATION=1。

int* TemplateMatch(image input,image Template,char* output,int channels,int ROTATION)	模板匹配，input 是母本图像，Template 是样本图像，output 是结果图像文件名。channels 是图像的像素通道数,返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的 X 轴坐标值，第三个元素是目标的 Y 轴坐标值。至少支持 PNG 图像。参考：channels=3，ROTATION=360 或 ROTATION=1。
	需引入以下头文件和结构体：
	#include "stb_image.h"
	#include "stb_image_write.h"
	typedef struct imageContainer {
	int x,y,n;
	unsigned char *data;
	} image;
	参考：
	image input, Template;
	input.data = stbi_load(inputFile, &input.x, &input.y, &input.n, 3); //3 表示图像有 3 个像素通道
	Template.data = stbi_load(templateFile, &Template.x, &Template.y, &Template.n, 3);

int* TemplateMatch(char* input,char* Template,int channels,int	模板匹配，input 是母本图像，Template 是样本图像。channels 是图像的像素通道数，返回值数组中的第一个元素是最大匹配分数,第二个元素是目标的 X 轴坐标值，第三个元素是目标的 Y 轴坐标值。至少支持 PNG 图像。参考：channels=3，ROTATION=360 或

ROTATION)	ROTATION=1。

int* TemplateMatch(image input,image Template,int ROTATION)	模板匹配，input 是母本图像，Template 是样本图像。返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的 X 轴坐标值,第三个元素是目标的 Y 轴坐标值。至少支持 PNG 图像。参考：ROTATION=360 或 ROTATION=1。
	需引入以下头文件和结构体：
	#include "stb_image.h"
	#include "stb_image_write.h"
	typedef struct imageContainer {
	int x,y,n;
	unsigned char *data;
	} image;
	参考：
	image input, Template;
	input.data = stbi_load(inputFile, &input.x, &input.y, &input.n, 3); //3 表示图像有 3 个像素通道
	Template.data = stbi_load(templateFile, &Template.x, &Template.y, &Template.n, 3);

void PGMSobel(char* input,char* output,int Mx[3][3],int My[3][3],int max,int min)	Sobel 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 文件。 参考模板： int Mx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}} int My[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}} int max = -9999 int min = 9999
void PGMSobelX(char* input,char* output,int Mx[3][3],int My[3][3],int max,int min)	X 方向滤波，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 文件。 参考模板： int Mx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}} int My[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}} int max = -9999 int min = 9999
void PGMSobelY(char* input,char* output,int Mx[3][3],int My[3][3],int max,int min)	Y 方向滤波，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 文件。 参考模板： int Mx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}} int My[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1,

	2, 1}} int max = -9999 int min = 9999

void PGMSobel1(char* input,char* output,int min,int max,int mx[3][3],int my[3][3])	Sobel 算子，input 是输入文件名，output 是输出文件名。min 和 max 是图像归一化相关参数，如 min = 1000000, max = 0；mx 和 my 分别是 Sobel 算子的 X 和 Y 方向模板。支持 P2 和 P5 格式的 PGM。
	参考模板 ：
	int mx[3][3] = {
	{-1, 0, 1},
	{-2, 0, 2},
	{-1, 0, 1}
	};
	int my[3][3] = {
	{-1, -2, -1},
	{0, 0, 0},
	{1, 2, 1}
	};

void PGMSobelX1(char* input,char* output,int min,int max,int mx[3][3],int my[3][3])	X 方向梯度，input 是输入文件名，output 是输出文件名。min 和 max 是图像归一化相关参数，如 min = 1000000, max = 0；mx 和 my 分别是 Sobel 算子的 X 和 Y 方向模板。支持 P2 和 P5 格式的 PGM。
	参考模板 ：
	int mx[3][3] = {
	{-1, 0, 1},
	{-2, 0, 2},
	{-1, 0, 1}
	};
	int my[3][3] = {
	{-1, -2, -1},
	{0, 0, 0},
	{1, 2, 1}
	};

void PGMSobelY1(char* input,char* output,int min,int max,int mx[3][3],int my[3][3])	Y 方向梯度，input 是输入文件名，output 是输出文件名。min 和 max 是图像归一化相关参数，如 min = 1000000, max = 0；mx 和 my 分别是 Sobel 算子的 X 和 Y 方向模板。支持 P2 和 P5 格式的 PGM。
	参考模板 ：
	int mx[3][3] = {
	{-1, 0, 1},
	{-2, 0, 2},
	{-1, 0, 1}
	};
	int my[3][3] = {

	{-1, -2, -1}, {0, 0, 0}, {1, 2, 1} };
void PGMSobel2(char* input,char* XOutput,char* YOutput,char* SobelOutput,int sobel_x[3][3],int sobel_y[3][3],int min,int max)	Sobel 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。XOutput 是输出的 X 方向的梯度图像，YOutput 是输出的 Y 方向的梯度图像，SobelOutput 是输出的整幅图像的 Sobel 算子计算结果，min 和 max 是图像归一化的相关参数，如 min=100,max=0。 参考模板： int sobel_x[3][3]={{-1, 0, 1},{-2, 0, 2},{-1, 0, 1}}; int sobel_y[3][3]={{1, 2, 1}, {0, 0, 0},{-1,-2,-1}};
void Sobel(char* input,char* output)	Sobel 算子，input 是输入文件名，output 是输出文件名。支持 PGM 文件。
void Laplatian(char* input,char* output)	Laplatian 算子，input 是输入文件名，output 是输出文件名。支持 PGM 文件。
void HorizSobel(char* input,char* output)	水平 Sobel 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。
void VertSobel(char* input,char* output)	垂直 Sobel 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。
void PGMSobel1(char* input,char* output,int threshold)	Sobel 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。threshold 是目标阈值，如 threshold=80。
void YFiltering(char* input,char* output,int sobel_x[3][3],int sobel_y[3][3])	Y 方向滤波，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。 参考模板： int sobel_x[3][3] = { { 1, 0, -1}, { 2, 0, -2}, { 1, 0, -1}}; int sobel_y[3][3] = { { 1, 2, 1}, { 0, 0, 0}, {-1, -2, -1}};
void XFiltering(char* input,char* output,int sobel_x[3][3],int sobel_y[3][3])	X 方向滤波，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。 参考模板： int sobel_x[3][3] = { { 1, 0, -1}, { 2, 0, -2}, { 1, 0, -1}}; int sobel_y[3][3] = { { 1, 2, 1},

	{ 0, 0, 0}, {-1, -2, -1}};
void SobelFiltering(char* input,char* output,int sobel_x[3][3],int sobel_y[3][3])	Sobel 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。 参考模板： int sobel_x[3][3] = { { 1, 0, -1}, { 2, 0, -2}, { 1, 0, -1}}; int sobel_y[3][3] = { { 1, 2, 1}, { 0, 0, 0}, {-1, -2, -1}};
void PrewittFiltering(char* input,char* output,int prewitt_x[3][3],int prewitt_y[3][3])	Prewitt 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。 参考模板： int prewitt_x[3][3] = { { 5, 5, 5}, { -3, 0, -3}, { -3, -3, -3}}; int prewitt_y[3][3] = { { 5, -3, -3}, { 5, 0, -3}, {5, -3, -3}};
void LaplacianFiltering(char* input,char* output,int laplacian[3][3])	Laplace 算子，input 是输入文件名，output 是输出文件名。支持 P5 格式的 PGM 图像。laplacian 是 Laplacian 算子模板。 参考模板： int laplacian[3][3] = { { 1, 1, 1}, { 1, -8, 1}, { 1, 1, 1}};
vector<float> HarrisCornerDetection(char* input,int width,int height,int channels,int step,float threshold,float k,float sigma)	支持 P5 和 P6 格式的 PPM 文件，从返回值数组中的第一个元素开始，返回值以 3 个元素为一组，分别是角点的 X 坐标、Y 坐标和分数，若返回值数组名为 A，则 {A[0],A[1],A[2]} 是第一个角点的数据，{A[3],A[4],A[5]}是第二个角点的数据，以此类推。input 是输入的图像文件名，width 和 height 是输入图像的宽和高，channels 是输入图像的通道数，step 默认=-1，threshold 是 Harris 检测中角点的得分阈值，k 是 Harris 评分函数中的 k 值，sigma 是用于 IxIy 阵列平滑的 sigma 值，参考：threshold=2000，k=1，sigma=1.2。
vector<float> HarrisCorner(char* input,char* output,int width,int height,int channels,float	支持 P5 和 P6 格式的 PPM 文件，从返回值数组中的第一个元素开始，返回值以 3 个元素为一组，分别是角点的 X 坐标、Y 坐标和分数，若返回值数组名为 A，则 {A[0],A[1],A[2]} 是第一个角点的数据，{A[3],A[4],A[5]}是第二个角点的数据，以此类推。input 是输入的图像文件名，width 和 height 是输

threshold,float k,float sigma)	入图像的宽和高，channels 是输入图像的通道数，threshold 是 Harris 检测中角点的得分阈值，k 是 Harris 评分函数中的 k 值，sigma 是用于 IxIy 阵列平滑的 sigma 值，参考：threshold=2000，k=1，sigma=1.2。
double* TemplateMatching1(Image2* input, Image2* Template,char* output,char* output_txt,double threshold,int isWriteImageResult,unsigned char color,unsigned char red,unsigned char green,unsigned char blue)	模板匹配，返回值数组：匹配框左上角顶点的 X 和 Y 坐标、模板的宽和高、差异度。参考：output 是匹配结果图像名，output_txt 是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color 是当图像是灰度图时的匹配框颜色，red、green 和 blue 是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持 PPM 文件。 需要引入以下结构体： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; 声明： Image2* readPXM(const char* name); 参考： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
double* TemplateMatching1(Image2* input, Image2* Template,char* output_txt,double threshold,unsigned char color,unsigned char red,unsigned char green,unsigned char blue)	模板匹配，返回值数组：匹配框左上角顶点的 X 和 Y 坐标、模板的宽和高、差异度。参考：output_txt 是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color 是当图像是灰度图时的匹配框颜色，red、green 和 blue 是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持 PPM 文件。 需要引入以下结构体： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; 声明： Image2* readPXM(const char* name); 参考： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);

double* TemplateMatching2(Image2* input, Image2* Template,char* output,char* output_txt,double threshold,int isWriteImageResult,unsigned char color,unsigned char red,unsigned char green,unsigned char blue)	模板匹配，返回值数组：匹配框左上角顶点的 X 和 Y 坐标、模板的宽和高、差异度。参考：output 是匹配结果图像名，output_txt 是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color 是当图像是灰度图时的匹配框颜色，red、green 和 blue 是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持 PPM 文件。 需要引入以下结构体： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; 声明： Image2* readPXM(const char* name); 参考： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
double* TemplateMatching2(Image2* input, Image2* Template,char* output_txt,double threshold,unsigned char color,unsigned char red,unsigned char green,unsigned char blue)	模板匹配，返回值数组：匹配框左上角顶点的 X 和 Y 坐标、模板的宽和高、差异度。参考：output_txt 是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color 是当图像是灰度图时的匹配框颜色，red、green 和 blue 是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持 PPM 文件。 需要引入以下结构体： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; 声明： Image2* readPXM(const char* name); 参考： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
Image2* TemplateMatching3(Image2* input, Image2* Template, char* output_txt, double	模板匹配，返回匹配结果的图像数据。参考：output_txt 是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color 是当图像是灰度图时的匹配框颜色，red、green 和 blue 是当图像是彩色图时的匹配框颜色的红绿蓝通道值。

threshold, int isWriteImageResult, unsigned char color, unsigned char red, unsigned char green, unsigned char blue)	需要引入以下结构体： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; 声明： Image2* readPXM(const char* name); 参考： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
Image2* TemplateMatching4(Image2* input, Image2* Template, char* output_txt, double threshold, int isWriteImageResult, unsigned char color, unsigned char red, unsigned char green, unsigned char blue)	图像匹配，返回匹配结果的图像数据。参考：output_txt 是保存的匹配相关数据的文本文件，threshold=0.5,isWriteImageResult=1,color 是当图像是灰度图时的匹配框颜色，red、green 和 blue 是当图像是彩色图时的匹配框颜色的红绿蓝通道值。 需要引入以下结构体： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; 声明： Image2* readPXM(const char* name); 参考： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
double* TemplateMatching(RGB_PACKED_IMAGE* input, RGB_PACKED_IMAGE* Template,char* output,unsigned char red,unsigned char green,unsigned char blue,double c,double threshold)	模板匹配，返回匹配框的中心点坐标、旋转角度和缩放比例。参考：c=0.5，threshold=0.9。支持 PPM 文件。 声明： #ifndef __P #if defined(__STDC__) || defined(__cplusplus) #define __P(protos) protos #else #define __P(protos) () #endif #endif RGB_PACKED_IMAGE *readRGBPackedImage __P((char*));

	需引入以下结构体：
	typedef struct rgb_packed_pixel {
	BYTE r;
	BYTE g;
	BYTE b;
	} RGB_PACKED_PIXEL;
	typedef struct rgb_packed_image {
	int cols;
	int rows;
	RGB_PACKED_PIXEL **p;
	RGB_PACKED_PIXEL *data_p;
	} RGB_PACKED_IMAGE;
	参考：
	RGB_PACKED_IMAGE* Template = readRGBPackedImage(templatename);
	RGB_PACKED_IMAGE* input = readRGBPackedImage(inputFileName);

double* TemplateMatching1(RGB_PACKED_IMAGE* input, RGB_PACKED_IMAGE* Template,unsigned char red,unsigned char green,unsigned char blue,double c,double threshold)	模板匹配，返回匹配框的中心点坐标、旋转角度和缩放比例。参考：c=0.5，threshold=0.9。支持 PPM 文件。
	声明：
	#ifndef __P
	#if defined(__STDC__) || defined(__cplusplus)
	#define __P(protos) protos
	#else
	#define __P(protos) ()
	#endif
	#endif
	RGB_PACKED_IMAGE *readRGBPackedImage __P((char*));
	需引入以下结构体：
	typedef struct rgb_packed_pixel {
	BYTE r;
	BYTE g;
	BYTE b;
	} RGB_PACKED_PIXEL;
	typedef struct rgb_packed_image {
	int cols;
	int rows;
	RGB_PACKED_PIXEL **p;
	RGB_PACKED_PIXEL *data_p;
	} RGB_PACKED_IMAGE;
	参考：
	RGB_PACKED_IMAGE* Template =

	readRGBPackedImage(templatename); RGB_PACKED_IMAGE* input = readRGBPackedImage(inputFileName);

double* TemplateMatching2(RGB_PACKED_IMAGE* input, RGB_PACKED_IMAGE* Template,unsigned char red,unsigned char green,unsigned char blue,double c,double threshold)	模板匹配，返回匹配框的中心点坐标、旋转角度和缩放比例。参考：c=0.5，threshold=0.9。支持 PPM 文件。
	声明：
	#ifndef __P
	#if defined(__STDC__) || defined(__cplusplus)
	#define __P(protos) protos
	#else
	#define __P(protos) ()
	#endif
	#endif
	RGB_PACKED_IMAGE *readRGBPackedImage __P((char*));
	需引入以下结构体：
	typedef struct rgb_packed_pixel {
	BYTE r;
	BYTE g;
	BYTE b;
	} RGB_PACKED_PIXEL;
	typedef struct rgb_packed_image {
	int cols;
	int rows;
	RGB_PACKED_PIXEL **p;
	RGB_PACKED_PIXEL *data_p;
	} RGB_PACKED_IMAGE;
	参考：
	RGB_PACKED_IMAGE* Template = readRGBPackedImage(templatename);
	RGB_PACKED_IMAGE* input = readRGBPackedImage(inputFileName);

RGB_PACKED_IMAGE* TemplateMatching(RGB_PACKED_IMAGE* input, RGB_PACKED_IMAGE* Template,unsigned char red, unsigned char green, unsigned char blue, double c, double threshold)	模板匹配，返回匹配结果的图像数据。参考：c=0.5，threshold=0.9。支持 PPM 文件。 声明： #ifndef __P #if defined(__STDC__) || defined(__cplusplus) #define __P(protos) protos #else #define __P(protos) () #endif #endif RGB_PACKED_IMAGE *readRGBPackedImage

	__P((char*));
	需引入以下结构体：
	typedef struct rgb_packed_pixel {
	BYTE r;
	BYTE g;
	BYTE b;
	} RGB_PACKED_PIXEL;
	typedef struct rgb_packed_image {
	int cols;
	int rows;
	RGB_PACKED_PIXEL **p;
	RGB_PACKED_PIXEL *data_p;
	} RGB_PACKED_IMAGE;
	参考：
	RGB_PACKED_IMAGE* Template = readRGBPackedImage(templatename);
	RGB_PACKED_IMAGE* input = readRGBPackedImage(inputFileName);

unsigned int FeatureDetection(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue)	特征检测，返回特征点数量。支持 PPM 文件。
vector<int> FeatureMatching(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue)	特征匹配，返回值中的数据格式为：第 1 个值是特征对的序号，从 0 开始，第 2 个值和第 3 个值分别是是图像 1 的其中一个特征点的横坐标和纵坐标，第 4 个值和第 5 个值分别是是图像 2 的其中一个特征点的横坐标和纵坐标并与第 2 个值和第 3 个值所在的图像 1 的特征点相对应，这 5 个值构成一组；同理，第 6 个值就是下一个特征对的序号，即为 1，后面的值则以此类推。格式如：特征对：%d，图像 1(%d, %d)->图像 2：(%d, %d)。支持 PPM 文件。
unsigned int FeatureMatching1(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	从两幅输入图像中检测特征点，然后使用蛮力方法匹配两幅图像的特征。input 是输入图像，output 是生成的特征点图像，bExtractDescriptor=true。返回匹配的特征点数量。支持 PGM 文件。
vector<int>	返回值中的数据格式为：第 1 个值是特征对的序号，

FeatureMatching2(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	从 0 开始，第 2 个值和第 3 个值分别是是图像 1 的其中一个特征点的横坐标和纵坐标，第 4 个值和第 5 个值分别是是图像 2 的其中一个特征点的横坐标和纵坐标并与第 2 个值和第 3 个值所在的图像 1 的特征点相对应，这 5 个值构成一组；同理，第 6 个值就是下一个特征对的序号，即为 1，后面的值则以此类推。格式如：特征对：%d，图像 1(%d, %d)->图像 2：(%d, %d)。input 是输入图像，output 是生成的特征点图像，bExtractDescriptor=true。支持 PGM 文件。
unsigned int FeatureExtraction1(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	返回特征点数量。input 是输入图像，output 是生成的特征点图像，bExtractDescriptor=true。支持 PGM 文件。

std::list<ezsift::SiftKeypoint> FeatureExtraction2(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	返回特征点列表。input 是输入图像，output 是生成的特征点图像，bExtractDescriptor=true。支持 PGM 文件。
	需引入以下命名空间：
	namespace ezsift {
	#define DEGREE (128) //SIFT 关键点: 128 维
	struct SiftKeypoint {
	int octave; // octave 数量
	int layer; // layer 数量
	float rlayer; // layer 实际数量
	float r; // 归一化的 row 坐标
	float c; // 归一化的 col 坐标
	float scale; // 归一化的 scale
	float ri; // row 坐标(layer)
	float ci; // column 坐标(layer)
	float layer_scale; // scale(layer)
	float ori; // 方向(degrees)
	float mag; // 模值
	float descriptors[DEGREE]; //描述符
	};
	}

vector<int> DefectLocation(PGMData* Template,PGMData* Sample,int floor,int size,int a,int b,int	查找缺陷位置，返回材料缺陷的位置，以每 4 个元素为一组。Template 是模板图像，Sample 是样本图像，g 是缺陷的有效界限的 X 轴长度，h 是缺陷的有效界限的 Y 轴长度，参考：floor=80,size=10,a=64,b=64,c=16,d=16,e=2,f=4,g=65,h=65,FULL=0,EMPTY=255,report=true。

c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report)	引入以下结构体：
	typedef struct _PGMData {
	int row;
	int col;
	int max_gray;
	int **matrix;
	}PGMData;
	若模板文件名为 Template，样本文件名为 Sample，使用以下代码获得合适的输入数据：
	首先声明 readPGM 函数：
	PGMData* readPGM(const char *file_name, PGMData *data);
	然后执行以下代码：
	PGMData* model = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Template,model);
	PGMData* data = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Sample,data);
	之后将 model 和 data 传入相应的函数。
	支持 P5 格式的 PGM 文件。

vector<int> DefectSize(PGMData* Template,PGMData* Sample,int floor,int size,int a,int b,int c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report)	缺陷尺寸，返回缺陷尺寸，以 4 个为一组。Template 是模板图像，Sample 是样本图像，g 是缺陷的有效界限的 X 轴长度，h 是缺陷的有效界限的 Y 轴长度，参考：floor=80，size=10，a=64,b=64，c=16,d=16，e=2，f=4，g=65，h=65，FULL=0,EMPTY=255，report=true。
	引入以下结构体：
	typedef struct _PGMData {
	int row;
	int col;
	int max_gray;
	int **matrix;
	}PGMData;
	若模板文件名为 Template，样本文件名为 Sample，使用以下代码获得合适的输入数据：
	首先声明 readPGM 函数：
	PGMData* readPGM(const char *file_name, PGMData *data);
	然后执行以下代码：
	PGMData* model = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Template,model);
	PGMData* data =

	(PGMData*)malloc(sizeof(PGMData)); readPGM(Sample,data); 之后将 model 和 data 传入相应的函数。 支持 P5 格式的 PGM 文件。

vector<int> GoodBadQuantity(PGMData* Template,PGMData* Sample,int floor,int size,int a,int b,int c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report)	样品好坏的数量，返回结果中，第一个元素是合格的圆圈数量，第二个元素是缺陷的圆圈数量。Template 是模板图像，Sample 是样本图像，g 是缺陷的有效界限的 X 轴长度，h 是缺陷的有效界限的 Y 轴长度，参考：floor=80，size=10，a=64,b=64，c=16,d=16，e=2，f=4，g=65，h=65，FULL=0,EMPTY=255，report=true。
	引入以下结构体：
	typedef struct _PGMData {
	int row;
	int col;
	int max_gray;
	int **matrix;
	}PGMData;
	若模板文件名为 Template，样本文件名为 Sample，使用以下代码获得合适的输入数据：
	首先声明 readPGM 函数：
	PGMData* readPGM(const char *file_name, PGMData *data);
	然后执行以下代码：
	PGMData* model = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Template,model);
	PGMData* data = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Sample,data);
	之后将 model 和 data 传入相应的函数。
	支持 P5 格式的 PGM 文件。

void RAWSobelEdge(char* input,char* output,int ROWS,int COLS,int M,float sobelX[3][3],float sobelY[3][3])	Sobel 算子，input 是输入文件名，output 是输出文件名。ROWS 是图像的行，COLS 是图像的列，M 是滤波相关参数，如 M=1。支持 RAW 图像。 参考模板： float sobelX[3][3] = {{-1,0,1}, {-2,0,2}, {-1,0,1}}; float sobelY[3][3] = {{-1,-2,-1}, {0,0,0}, {1,2,1}};
void	边缘检测，input 是输入文件名，output 是输出文件

RAWPlaceHolder(char* input,char* output,int ROWS,int COLS,int M,float mask[3][3])	名。ROWS 是图像的行，COLS 是图像的列，M 是滤波相关参数，如 M=1。支持 RAW 图像。 参考模板： float mask[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};
void RAWLaplacialSharpeningFilter(char* input,char* output,int ROWS,int COLS,int M,float w,float mask[3][3])	拉普拉斯锐化滤波器，input 是输入文件名，output 是输出文件名。ROWS 是图像的行大小，COLS 是图像的列大小，M 和 w 是滤波相关参数，如 M=1，w=1；mask 是滤波器模板。支持 RAW 图像。 参考模板： float mask[3][3] = {{0,1,0}, {1,-4,1}, {0,1,0}};
void RawLaplacianEnhancement(char* input1,char* output1,int width,int height)	拉普拉斯算子增强，input1 是输入的 RAW 图像文件名，output1 是输出的 RAW 图像文件名，width 是输入图像的宽，height 是输入图像的高。支持 RAW 图像。

struct hough_param_circle* CircleDetection(char* input,int width,int height)	圆检测，返回找到的圆的位置和大小等相关信息。支持 RAW 文件。
	需引入以下结构体：
	struct hough_param_circle {
	int a;
	int b;
	int radius;
	int resolution;
	int thresh;
	struct point *points;
	int points_size;
	};

void ImageWarpEllipticalGrid(string input,string output,int height,int width,int NumberChannels)	圆样变形，支持 RAW 文件。
int* FindLine(char* input,char* output,int width,int height)	直线检测，返回直线的 theta 和 rho，支持 RAW 图像。
int* FindCircle(char*	圆检测，返回圆心的坐标和圆的半径，支持 RAW 图像。sigma=1.4，tmin=70，tmax=150。

input,char* output,int width,int height,float sigma, int tmin, int tmax)	
void CornerDetection(char* input,char* output,float threshold, float k, float sigma,int width, int height, int channels)	角点检测，threshold=10000，k=0.06，sigma=1.0，width=640，height=480，channels=1。支持 PNM 图像。
vector<Keypoint> CornerDetection1(char* input,char* output,float threshold, float k, float sigma,int width, int height, int channels)	角点检测,返回角点数据。threshold=10000,k=0.06，sigma=1.0，width=640，height=480，channels=1。支持 PNM 图像。 需引入以下结构体 typedef struct { float x; float y; float score; } Keypoint;

vector<Corner::Keypoint> CornerDetection(char* input,int width,int height,int channels,float threshold, float k, float sigma)	角点检测，返回角点数据。threshold=2000，k=1，sigma=1.2。支持 PNM 图像。
	需引入以下命名空间：
	namespace Corner {
	struct Keypoint {
	float x;
	float y;
	float score;
	};
	}

void Structure(char* input,char* output,float sigma)	特征归一化统计，参考：sigma=2。支持多种图像格式。
void Cornerness(char* input,char* output,float sigma,int method)	角点检测，参考：sigma=2，method=0。支持多种图像格式。
void Corners(char* input,char* output,float sigma, float thresh, int window, int nms, int	角点检测，参考：sigma=2，thresh=0.4，window=5，nms=3，corner_method=0。支持多种图像格式。

corner_method)	
void FindMatch(char* input1,char* input2,char* output,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner_method,float sigma1, float thresh1, int window1, int nms1, int corner_method1,float sigma2, float thresh2, int window2, int nms2, int corner_method2,float sigma5, float thresh5, int window5, int nms5, int corner_method5,float sigma6, int corner_method6, float thresh6, int window6, int nms6, float inlier_thresh6, int iters6, int cutoff6, float acoeff6)	特征匹配，参考：thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner_method=0 ， sigma1=2 ， thresh1=0.4 ，window1=5，nms1=3，corner_method1=0，sigma2=2，thresh2=0.4 ， window2=5 ， nms2=3 ，corner_method2=0 ， sigma5=2 ， thresh5=0.4 ，window5=7，nms5=3，corner_method5=0，sigma6=2，corner_method6=0 ， thresh6=0.3 ， window6=7 ，nms6=3 ， inlier_thresh6=5 ， iters6=1000 ，cutoff6=50，acoeff6=0.5。支持多种图像格式。

vector<Descriptor> HarrisCorner(char* input,char* output,float sigma1, float thresh1, int window1, int nms1, int corner_method1)	角点检测，返回检测结果。参考：sigma1=2，thresh1=0.4 ， window1=5 ， nms1=3 ，corner_method1=0。支持多种图像格式。
	需引入以下结构体：
	struct Point {
	double x, y;
	
	Point() : x(0), y(0) {}
	Point(double x, double y) : x(x), y(y) {}
	};

	struct Descriptor {
	  Point p;
	  vector<float> data;
	
	  Descriptor(){}
	  Descriptor(const Point& p) : p(p) {}
	};

vector<Match> MatchDescriptors(char* input1,char* input2,char* output,float sigma1, float thresh1, int window1, int nms1, int corner_method1,float sigma2, float thresh2, int window2, int nms2, int corner_method2)	描述匹配项，返回描述结果。参考：sigma1=2，thresh1=0.4 ， window1=5 ， nms1=3 ，corner_method1=0 ， sigma2=2 ， thresh2=0.4 ，window2=5，nms2=3，corner_method2=0。支持多种图像格式。
	需引入以下结构体：
	struct Point {
	  double x, y;
	
	  Point() : x(0), y(0) {}
	  Point(double x, double y) : x(x), y(y) {}
	};
	struct Descriptor {
	  Point p;
	  vector<float> data;
	
	  Descriptor(){}
	  Descriptor(const Point& p) : p(p) {}
	};
	struct Match {
	  const Descriptor* a=nullptr;
	  const Descriptor* b=nullptr;
	  float distance=0.f;
	
	  Match(){}
	  Match(const Descriptor* a,const Descriptor* b,float dist=0.f) : a(a), b(b), distance(dist) {}
	
	  bool operator<(const Match& other) { return distance<other.distance; }
	};

void DrawInliers(char* input1,char* input2,char* output,float	绘制角点，参考：thresh3=5，k=10000，cutoff=50，thresh4=5,sigma=2,thresh=0.4,window=5,nms=3,corner_method=0 ， sigma1=2 ， thresh1=0.4 ，window1=5，nms1=3，corner_method1=0，sigma2=2，thresh2=0.4 ， window2=5 ， nms2=3 ，

thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner_method,float sigma1, float thresh1, int window1, int nms1, int corner_method1,float sigma2, float thresh2, int window2, int nms2, int corner_method2,float sigma5, float thresh5, int window5, int nms5, int corner_method5,float sigma6, int corner_method6, float thresh6, int window6, int nms6, float inlier_thresh6, int iters6, int cutoff6, float acoeff6)	corner_method2=0， sigma5=2， thresh5=0.4，window5=7，nms5=3，corner_method5=0，sigma6=2，corner_method6=0， thresh6=0.3， window6=7，nms6=3， inlier_thresh6=5， iters6=1000，cutoff6=50，acoeff6=0.5。支持多种图像格式。
void PanoramaImage(char* input1,char* input2,char* output,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner_method,float sigma1, float thresh1, int window1, int nms1, int	制造全景图像，参考：thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner_method1=0， sigma2=2， thresh2=0.4，window2=5，nms2=3，corner_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner_method5=0，sigma6=2，corner_method6=0，thresh6=0.3，window6=7，nms6=3，inlier_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5。支持多种图像格式。

corner_method1,float sigma2, float thresh2, int window2, int nms2, int corner_method2,float sigma5, float thresh5, int window5, int nms5, int corner_method5,float sigma6, int corner_method6, float thresh6, int window6, int nms6, float inlier_thresh6, int iters6, int cutoff6, float acoeff6)	
void Cylindrical(char* input1,char* input2,char* output,float f1,float f2,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner_method,float sigma1, float thresh1, int window1, int nms1, int corner_method1,float sigma2, float thresh2, int window2, int nms2, int corner_method2,float sigma5, float thresh5, int window5, int nms5,	角点检测。参考：f1=500，f2=500，thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner_method5=0，sigma6=2，corner_method6=0，thresh6=0.3，window6=7，nms6=3，inlier_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5，sigma7=2，corner_method7=0，thresh7=0.3，window7=7，nms7=3，inlier_thresh7=5，iters7=1000，cutoff7=50，acoeff7=0.5。支持多种图像格式。

int corner_method5,float sigma6, int corner_method6, float thresh6, int window6, int nms6, float inlier_thresh6, int iters6, int cutoff6, float acoeff6,float sigma7, int corner_method7, float thresh7, int window7, int nms7, float inlier_thresh7, int iters7, int cutoff7, float acoeff7)	
void Spherical(char* input1,char* input2,char* output,float f1,float f2,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner_method,float sigma1, float thresh1, int window1, int nms1, int corner_method1,float sigma2, float thresh2, int window2, int nms2, int corner_method2,float sigma5, float thresh5, int window5, int nms5, int corner_method5,float	角点检测。参考：f1=500，f2=500，thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner_method5=0，sigma6=2，corner_method6=0，thresh6=0.3，window6=7，nms6=3，inlier_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5，sigma7=2，corner_method7=0，thresh7=0.3，window7=7，nms7=3，inlier_thresh7=5，iters7=1000，cutoff7=50，acoeff7=0.5。支持多种图像格式。

sigma6, int corner_method6, float thresh6, int window6, int nms6, float inlier_thresh6, int iters6, int cutoff6, float acoeff6,float sigma7, int corner_method7, float thresh7, int window7, int nms7, float inlier_thresh7, int iters7, int cutoff7, float acoeff7)	
void FeatureMatching(char* input1,char* input2,char* output)	特征匹配，支持 PGM 图像。
int FeatureMatching(char* input1,char* input2,char* output,char* outputCouple)	特征匹配，返回匹配的点对，支持 BMP 图像。

std::vector <CentersPoint> FindCircles(char* input,char* output,int size1,int size2,int size3)	圆检测，返回检测结果。size1=5，size2=5，size3=7。支持 BMP 图像。
	需引入以下结构体：
	struct Point {
	Point(int x = 0, int y = 0) { this->x = x; this->y = y; }
	int x;
	int y;
	};
	struct CentersPoint {
	CentersPoint(Point point, int radius) { this->point = point; this->radius = radius; count = 1; }
	Point point;
	int count;
	int radius;
	};

void Canny(char*	Canny 算子，至少支持 JPG 图像，input 是输入文件

input,char* output,int lowThreshold,int highThreshold)	名，output 是输出文件名，参考：lowThreshold=50，highThreshold=150。
void KMeans1(char* input,char* output,int c,int k)	K-Means 聚类，input 是输入文件名，output 是输出文件名。输入图像最好宽高相同，c 的最大值是图像的宽和高中较小的那个参数，如宽=500，高为 600，则 c 最大可取 500；k 是聚类的种类数目。支持 BMP 图像。
void KMeans(string input,unsigned int Clusters,char* output)	K-Means 聚类，input 是输入文件名，Clusters 是聚类的种类数目，output 是输出文件名。支持 BMP 图像。
void LSBRead(char* input,char* output,int width,int height,unsigned char color1,unsigned char color2)	LSB 隐写文件的读出，input 是输入文件名，output 是输出文件名。参考：color1=255，color2=0。支持 24 位 BMP 图像。
void LSBWrite(char* input1,char* input2,char* output,int width,int height,unsigned char threshold1,unsigned char threshold2,unsigned char threshold3,unsigned char color1,unsigned char color2)	LSB 隐写，input1 是用于容纳隐藏图像的图像，input2 是要隐藏的黑白图像，参考：threshold1、threshold2 和 threshold3 都等于 128，color1=(unsigned char) 0b00000001，color2=(unsigned char) 0b11111110。支持 24 位 BMP 图像。
void Roberts(unsigned char** input,unsigned char** output)	Roberts 算子，input 是输入数据，output 是输出数据。
void Roberts(BMPMat** input,BMPMat** output)	Roberts 算子，input 是输入数据，output 是输出数据。
void STLSection(char* input,char* output,int	STL 切片，input 是输入的 STL 文件，output 是输出的切片文件前缀名，sliceAmount 是切片量，如：sliceAmount=50，resolution 是分辨率，如：resolution=260，c 是执行的相关参数，如：c=5。

sliceAmount, int resolution,int c)	
void SURF(char* input1,char* input2,char* output)	SURF 算子，input1 和 input2 是输入文件名，output 是输出文件名。支持 BMP 图像。
void SobelBinary(char* input,char* output,char filterH[9],char filterV[9])	二进制法 Sobel 算子，input1 和 input2 是输入文件名，output 是输出文件名。支持 BMP 图像。 参考： char filterH[9] = {-1, 0, 1, -2, 0, 2, -1, 0, 1}; char filterV[9] = {1, 2, 1, 0, 0, 0, -1, -2, -1};
void SobelOperator(char* input,char* output)	Sobel 算子，耗时较长，input 是输入文件名，output 是输出文件名。支持 24 位 BMP 图像。
SobelImage** SobelOperator(char* input)	返回处理后的各像素点的坐标和对应的像素值，若是边缘点则对应白色，否则对应黑色。支持 BMP 图像。 需引入以下结构体： typedef struct { int x; int y; unsigned char red; unsigned char green; unsigned char blue; }SobelImage;
void Sobel2(char* input,char* output,char filterH[9],char filterV[9])	Sobel 算子，input 是输入文件名，output 是输出文件名。支持 BMP 图像。 参考： char filterH[9] = {-1, 0, 1, -2, 0, 2, -1, 0, 1}; char filterV[9] = {1, 2, 1, 0, 0, 0, -1, -2, -1};
void EdgeDetection(char* input,char* output)	边缘检测，input 是输入文件名，output 是输出文件名。支持 4 位 BMP 图像。
void EdgeDetection1(char* input,char* output,short sharpen[3][3])	边缘检测，input 是输入文件名，output 是输出文件名。支持 8 位 BMP 图像。 参考模板： short sharpen[3][3] = {{1, 1, 1}, {1, -8, 1},

	{1, 1, 1}};
void EdgeDetection2(char* input,char* output,int a)	边缘检测，input 是输入文件名，output 是输出文件名。a 是用于设置图像像素的相关参数，如 a=3。支持 24 位 BMP 图像。
void EdgeDetection3(char* input,char* output,int a)	边缘检测，input 是输入文件名，output 是输出文件名。a 是用于设置图像像素的相关参数，如 a=3。支持 24 位 BMP 图像。
void EdgeDetection4(char* input,char* output,int a)	边缘检测，input 是输入文件名，output 是输出文件名。a 是用于设置图像像素的相关参数，如 a=3。支持 24 位 BMP 图像。
void Roberts(char* input,char* output)	Roberts 边缘检测。支持 BMP 图像。
void Prewitt(char* input,char* output)	Prewitt 边缘检测。支持 BMP 图像。
void Sobel(char* input,char* output)	Sobel 边缘检测。支持 BMP 图像。
void Laplace(char* input,char* output)	Laplace 边缘检测。支持 BMP 图像。
void QRCodeGeneration(char *filename, char* inputString)	二维码生成，filename 是生成的二维码图像文件名，inputString 是二维码包含的信息。
void QRCodeEncode(char* input,char* output)	二维码编码，输入文本文件，输出二维码的 PBM 图像。
void QRCodeDecode(char* input,char* output)	二维码解码，输入为二维码的 PBM 图像，输出为文本文件。
void TemplateMatching(char* input,char* Template,char* output,unsigned int b,double ps,double a,double a1,double b1,double c)	模板匹配，a=0.5，b=0，a1=0.5，b1=0.5，c=0.2，ps 是相似度，如 ps=0.5。支持 24 位 BMP 图像。
SearchResult TemplateMatching(char* input,char* Template)	模板匹配，返回目标位置的左上角坐标和相似值。支持 PNG 图像。 需引入以下结构体： struct SearchResult { int x, y;

	double value; };

SearchResult TemplateMatching(uint8_t* input,uint8_t* Template,int imgWidth,int imgHeight,int imgBpp,int patWidth,int patHeight,int patBpp)	模板匹配，返回目标位置的左上角坐标和相似值。
	需引入以下头文件：
	#define STB_IMAGE_IMPLEMENTATION
	#include "stb_image.h"
	#include <cstdint>
	#include <complex>
	#include <vector>
	参考例程：
	int imgWidth, imgHeight, imgBpp;
	int patWidth, patHeight, patBpp;
	uint8_t* input = stbi_load(inputfile, &imgWidth, &imgHeight, &imgBpp, 3);
	uint8_t* Template = stbi_load(Templatefile, &patWidth, &patHeight, &patBpp, 3);

void TemplateMatching1(char* input,char* Template,char* output,float ps,image_colors colors)	模板匹配，ps是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。 支持24位BMP图像。 需引入以下结构体： typedef struct { unsigned char R, G, B; } image_colors;
void TemplateMatching1(image input,image Template,char* output,float ps,image_colors colors)	模板匹配，ps是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。支持24位BMP图像。 需引入以下结构体和头文件以及声明： #include <stdint.h> typedef struct { uint16_t signature; uint32_t size; uint16_t reserved1; uint16_t reserved2; uint32_t offset; uint32_t bitmapinfo; int32_t width; int32_t height; uint16_t no_planes; uint16_t no_bits_pixel; uint32_t compression; uint32_t size_padding; int32_t x_pixel_meter; int32_t y_pixel_meter; uint32_t no_colors;

	uint32_t no_imp_colors;
	} image_header;
	typedef struct {
	unsigned char R, G, B;
	} image_colors;
	typedef struct {
	image_colors *pixels; // Imaginea in forma liniarizata
	image_header header; // Header-ul imaginii
	int32_t padding;
	} image;
	bool grayscale_image(char* path_to_image, char* path_to_grey);
	image load_image(char* path_to_image);
	参考例程：
	char *path_to_grey="gray.bmp";
	grayscale_image(inputfile, path_to_grey);
	image input, Template;
	input = load_image(path_to_grey);
	Template = load_image(Templatefile);

window TemplateMatching3(char* input,char* Template,float ps,image_colors colors)	模板匹配，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。支持 24 位 BMP 图像。 需引入以下头文件和结构体： #include <stdint.h> typedef struct { unsigned char R, G, B; } image_colors; typedef struct { uint32_t x, y; image_colors colors; double ps; } x0y; typedef struct { uint32_t width, height, matches; x0y *pos; } window;
window TemplateMatching3(image input,image Template,float ps,image_colors colors)	模板匹配，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。支持 24 位 BMP 图像。 需引入以下结构体和头文件以及声明： #include <stdint.h> typedef struct {

	uint16_t signature;
	uint32_t size;
	uint16_t reserved1;
	uint16_t reserved2;
	uint32_t offset;
	uint32_t bitmapinfo;
	int32_t width;
	int32_t height;
	uint16_t no_planes;
	uint16_t no_bits_pixel;
	uint32_t compression;
	uint32_t size_padding;
	int32_t x_pixel_meter;
	int32_t y_pixel_meter;
	uint32_t no_colors;
	uint32_t no_imp_colors;
	} image_header;
	typedef struct {
	unsigned char R, G, B;
	} image_colors;
	typedef struct {
	image_colors *pixels; // Imaginea in forma liniarizata
	image_header header; // Header-ul imaginii
	int32_t padding;
	} image;
	bool grayscale_image(char* path_to_image, char* path_to_grey);
	image load_image(char* path_to_image);
	参考例程：
	char *path_to_grey="gray.bmp";
	grayscale_image(inputfile, path_to_grey);
	image input, Template;
	input = load_image(path_to_grey);
	Template = load_image(Templatefile);

window* TemplateMatching5(char* input,char* Template,float ps,image_colors colors)	模板匹配，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。支持 24 位 BMP 图像。
window* TemplateMatching5(im	模板匹配，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。支持 24 位 BMP

<table>
<tr>
<td>age input,image Template,float ps,image_colors colors)</td>
<td>图像。
需引入以下结构体和头文件以及声明：
#include <stdint.h>
typedef struct {
uint16_t signature;
uint32_t size;
uint16_t reserved1;
uint16_t reserved2;
uint32_t offset;
uint32_t bitmapinfo;
int32_t width;
int32_t height;
uint16_t no_planes;
uint16_t no_bits_pixel;
uint32_t compression;
uint32_t size_padding;
int32_t x_pixel_meter;
int32_t y_pixel_meter;
uint32_t no_colors;
uint32_t no_imp_colors;
} image_header;
typedef struct {
unsigned char R, G, B;
} image_colors;
typedef struct {
image_colors *pixels; // Imaginea in forma liniarizata
image_header header; // Header-ul imaginii
int32_t padding;
} image;
bool grayscale_image(char* path_to_image, char* path_to_grey);
image load_image(char* path_to_image);
参考例程：
char *path_to_grey="gray.bmp";
grayscale_image(inputfile, path_to_grey);
image input, Template;
input = load_image(path_to_grey);
Template = load_image(Templatefile);</td>
</tr>
<tr>
<td>void TemplateMatching2(char* input,char* Template,char*</td>
<td>模板匹配，input 是输入的 8 位 BMP 图像，Template 是 24 位模板图像，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。</td>
</tr>
</table>

output,float ps,image_colors colors)	

void TemplateMatching2(image input,image Template,char* output,float ps,image_colors colors)	模板匹配，input 是输入的 8 位 BMP 图像，Template 是 24 位模板图像，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。
	需引入以下结构体和头文件以及声明：
	#include <stdint.h>
	typedef struct {
	uint16_t signature;
	uint32_t size;
	uint16_t reserved1;
	uint16_t reserved2;
	uint32_t offset;
	uint32_t bitmapinfo;
	int32_t width;
	int32_t height;
	uint16_t no_planes;
	uint16_t no_bits_pixel;
	uint32_t compression;
	uint32_t size_padding;
	int32_t x_pixel_meter;
	int32_t y_pixel_meter;
	uint32_t no_colors;
	uint32_t no_imp_colors;
	} image_header;
	typedef struct {
	unsigned char R, G, B;
	} image_colors;
	typedef struct {
	image_colors *pixels; // Imaginea in forma liniarizata
	image_header header; // Header-ul imaginii
	int32_t padding;
	} image;
	bool grayscale_image(char* path_to_image, char* path_to_grey);
	image load_image(char* path_to_image);
	参考例程：
	char *path_to_grey="gray.bmp";
	grayscale_image(inputfile, path_to_grey);
	image input, Template;
	input = load_image(path_to_grey);

	Template = load_image(Templatefile);
window TemplateMatching4(char* input,char* Template,char* output,float ps,image_colors colors)	模板匹配，input 是输入的 8 位 BMP 图像，Template 是 24 位模板图像，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。

window TemplateMatching4(image input,image Template,char* output,float ps,image_colors colors)	模板匹配，input 是输入的 8 位 BMP 图像，Template 是 24 位模板图像，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。
	需引入以下结构体和头文件以及声明：
	#include <stdint.h>
	typedef struct {
	uint16_t signature;
	uint32_t size;
	uint16_t reserved1;
	uint16_t reserved2;
	uint32_t offset;
	uint32_t bitmapinfo;
	int32_t width;
	int32_t height;
	uint16_t no_planes;
	uint16_t no_bits_pixel;
	uint32_t compression;
	uint32_t size_padding;
	int32_t x_pixel_meter;
	int32_t y_pixel_meter;
	uint32_t no_colors;
	uint32_t no_imp_colors;
	} image_header;
	typedef struct {
	unsigned char R, G, B;
	} image_colors;
	typedef struct {
	image_colors *pixels; // Imaginea in forma liniarizata
	image_header header; // Header-ul imaginii
	int32_t padding;
	} image;
	bool grayscale_image(char* path_to_image, char* path_to_grey);
	image load_image(char* path_to_image);

	参考例程：
	char *path_to_grey="gray.bmp";
	grayscale_image(inputfile, path_to_grey);
	image input, Template;
	input = load_image(path_to_grey);
	Template = load_image(Templatefile);

window* TemplateMatching6(char* input,char* Template,char* output,float ps,image_colors colors)	模板匹配，input 是输入的 8 位 BMP 图像，Template 是 24 位模板图像，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。

window* TemplateMatching6(image input,image Template,char* output,float ps,image_colors colors)	模板匹配，input 是输入的 8 位 BMP 图像，Template 是 24 位模板图像，ps 是相似度，ps=0.8，colors=(image_colors){255, 0, 0}。
	需引入以下结构体和头文件以及声明：
	#include <stdint.h>
	typedef struct {
	uint16_t signature;
	uint32_t size;
	uint16_t reserved1;
	uint16_t reserved2;
	uint32_t offset;
	uint32_t bitmapinfo;
	int32_t width;
	int32_t height;
	uint16_t no_planes;
	uint16_t no_bits_pixel;
	uint32_t compression;
	uint32_t size_padding;
	int32_t x_pixel_meter;
	int32_t y_pixel_meter;
	uint32_t no_colors;
	uint32_t no_imp_colors;
	} image_header;
	typedef struct {
	unsigned char R, G, B;
	} image_colors;
	typedef struct {
	image_colors *pixels; // Imaginea in forma liniarizata
	image_header header; // Header-ul imaginii

	int32_t padding;
	} image;
	bool grayscale_image(char* path_to_image, char* path_to_grey);
	image load_image(char* path_to_image);
	参考例程：
	char *path_to_grey="gray.bmp";
	grayscale_image(inputfile, path_to_grey);
	image input, Template;
	input = load_image(path_to_grey);
	Template = load_image(Templatefile);

int* TemplateMatching(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue,double MatchScore)	模板匹配,返回值中第一个元素是匹配框左上角的纵坐标，第二个元素是匹配框左下角的纵坐标，第三个元素是匹配框的左上角的横坐标,第四个元素是匹配框的右上角的横坐标。input1 是搜索图像，input2 是模板图像，output 是匹配结果图像，MatchScore=0.9。
int* TemplateMatching(char* input1,char* input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore)	模板匹配,返回值中第一个元素是匹配框左上角的纵坐标，第二个元素是匹配框左下角的纵坐标，第三个元素是匹配框的左上角的横坐标,第四个元素是匹配框的右上角的横坐标。input1 是搜索图像，input2 是模板图像，output 是匹配结果图像，MatchScore=0.9。

struct imagine TemplateMatching(struct imagine ColorSource,struct imagine input1,struct imagine input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore)	模板匹配，返回匹配结果的图像，input1 是搜索图像，input2 是模板图像，MatchScore=0.9。需引入以下结构体：
	typedef struct imagine {
	unsigned char *R,*G,*B,*header;
	int W, H, Wpad, size;
	};
	声明：
	void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie);
	struct imagine salvareBitmap (char *destinatieFisier);
	void afisare (struct imagine img, char *destinatieSalvare);
	将灰度应用于所有图像：

	struct imagine output;
	grayscale_image(inputImage1, input1_grayscale);
	grayscale_image(inputImage2, input2_grayscale);
	ColorSource=salvareBitmap(inputImage1); //inputImage1 是彩色原图
	input1=salvareBitmap(input1_grayscale);
	input2=salvareBitmap(input2_grayscale);
	
	output=TemplateMatching(ColorSource,input1,input2,red,green,blue,MatchScore);
	afisare(output,outputfile);

int* TemplateMatching(struct imagine input1,struct imagine input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore)	模板匹配,返回值中第一个元素是匹配框左上角的纵坐标，第二个元素是匹配框左下角的纵坐标，第三个元素是匹配框的左上角的横坐标,第四个元素是匹配框的右上角的横坐标。input1 是搜索图像，input2 是模板图像，output 是匹配结果图像，MatchScore=0.9。
	需引入以下结构体：
	typedef struct imagine {
	unsigned char *R,*G,*B,*header;
	int W, H, Wpad, size;
	};
	声明：
	void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie);
	struct imagine salvareBitmap (char *destinatieFisier);
	将灰度应用于所有图像：
	struct imagine input1,input2;
	grayscale_image(inputImage1, input1_grayscale);
	grayscale_image(inputImage2, input2_grayscale);
	input1=salvareBitmap(input1_grayscale);
	input2=salvareBitmap(input2_grayscale);

void TemplateMatching(char* input,char* templatename,char* output,unsigned int MaximumMatchingQuant	模板匹配，suprapunereMaxima 表示最大重叠率，参考:MaximumMatchingQuantity=10,MatchScore=0.8,suprapunereMaxima=0.2。

ity,double MatchScore,float suprapunereMaxima,unsigned char red,unsigned char green,unsigned char blue)	
int* TemplateMatching(char* input,char* Template)	模板匹配，支持 24 位 BMP 图像，返回匹配框的左上角坐标（x，y）。

int* TemplateMatching(bmp_img input,bmp_img Template)	模板匹配，支持 24 位 BMP 图像，返回匹配框的左上角坐标（x，y）。
	需引入以下结构体和声明：
	enum bmp_error
	{
	BMP_FILE_NOT_OPENED = -4,
	BMP_HEADER_NOT_INITIALIZED,
	BMP_INVALID_FILE,
	BMP_ERROR,
	BMP_OK = 0
	};
	typedef struct _bmp_img
	{
	bmp_header img_header;
	bmp_pixel **img_pixels;
	} bmp_img;
	声明：
	enum bmp_error bmp_img_read(bmp_img *, const char *);
	参考例程：
	bmp_img input, Template;
	bmp_img_read(&input, inputfile);
	bmp_img_read(&Template, outputfile);

int* TemplateMatching(char* input1,char* input2,char* output)	模板匹配，返回值是匹配框的左上角坐标（x，y）。
int* TemplateMatching1(char* input1,char* input2)	模板匹配，返回值是匹配框的左上角坐标（x，y）。
int* TemplateMatching2(ch	模板匹配，返回值是匹配框的左上角坐标（x，y）。

ar* input1,char* input2)	

my_image_comp* TemplateMatching(my_image_comp input,my_image_comp Template,int H,int length,float* hpf)	模板匹配，返回匹配结果。
	需引入以下结构体：
	struct my_image_comp {
	int width;
	int height;
	int stride;
	int border;
	float *handle;
	float *buf;
	my_image_comp()
	{ width = height = stride = border = 0; handle = buf = NULL; }
	~my_image_comp()
	{ if (handle != NULL) delete[] handle; }
	void init(int height, int width, int border)
	{
	this->width = width; this->height = height; this->border = border;
	stride = width + 2*border;
	if (handle != NULL)
	delete[] handle;
	handle = new float[stride*(height+2*border)];
	buf = handle + (border*stride) + border;
	}
	void perform_boundary_extension();
	};
	struct filt {
	float* centre;
	int length;
	};
	声明：
	int read_bmp(char* image, my_image_comp* input_comps, int* num_comps, int H);
	int write_bmp(my_image_comp* output_comps, char* dest);
	filt make_filter(int type);
	参考：
	my_image_comp input;
	my_image_comp Template;

	filt filter = make_filter(1);
	int length = filter.length;
	float* hpf = filter.centre;
	int H = (filter.length - 1) / 2;
	int num_comps = 1;
	read_bmp(inputfile, &input, &num_comps, 0);
	read_bmp(Templatefile, &Template, &num_comps, H);
	write_bmp(&input,outputfile);

int* TemplateMatching1(my_image_comp input,my_image_comp Template,int H,int length,float* hpf)	模板匹配，返回值是匹配框的左上角坐标（x，y）。
	需引入以下结构体：
	struct my_image_comp {
	int width;
	int height;
	int stride;
	int border;
	float *handle;
	float *buf;
	my_image_comp()
	{ width = height = stride = border = 0; handle = buf = NULL; }
	~my_image_comp()
	{ if (handle != NULL) delete[] handle; }
	void init(int height, int width, int border)
	{
	this->width = width; this->height = height; this->border = border;
	stride = width + 2*border;
	if (handle != NULL)
	delete[] handle;
	handle = new float[stride*(height+2*border)];
	buf = handle + (border*stride) + border;
	}
	void perform_boundary_extension();
	};
	struct filt {
	float* centre;
	int length;
	};
	声明：
	int read_bmp(char* image, my_image_comp*

	input_comps, int* num_comps, int H);
	int write_bmp(my_image_comp* output_comps, char* dest);
	filt make_filter(int type);
	参考：
	my_image_comp input;
	my_image_comp Template;
	filt filter = make_filter(1);
	int length = filter.length;
	float* hpf = filter.centre;
	int H = (filter.length - 1) / 2;
	int num_comps = 1;
	read_bmp(inputfile, &input, &num_comps, 0);
	read_bmp(Templatefile, &Template, &num_comps, H);
	write_bmp(&input,outputfile);

int* TemplateMatching2(my_image_comp input,my_image_comp Template,int H,int length,float* hpf)	模板匹配，返回值是匹配框的左上角坐标（x，y）。
	需引入以下结构体：
	struct my_image_comp {
	int width;
	int height;
	int stride;
	int border;
	float *handle;
	float *buf;
	my_image_comp()
	{ width = height = stride = border = 0; handle = buf = NULL; }
	~my_image_comp()
	{ if (handle != NULL) delete[] handle; }
	void init(int height, int width, int border)
	{
	this->width = width; this->height = height; this->border = border;
	stride = width + 2*border;
	if (handle != NULL)
	delete[] handle;
	handle = new float[stride*(height+2*border)];
	buf = handle + (border*stride) + border;
	}
	void perform_boundary_extension();

	};
	struct filt {
	float* centre;
	int length;
	};
	声明：
	int read_bmp(char* image, my_image_comp* input_comps, int* num_comps, int H);
	int write_bmp(my_image_comp* output_comps, char* dest);
	filt make_filter(int type);
	参考：
	my_image_comp input;
	my_image_comp Template;
	filt filter = make_filter(1);
	int length = filter.length;
	float* hpf = filter.centre;
	int H = (filter.length - 1) / 2;
	int num_comps = 1;
	read_bmp(inputfile, &input, &num_comps, 0);
	read_bmp(Templatefile, &Template, &num_comps, H);
	write_bmp(&input,outputfile);

int* TemplateMatching(char* input1,char* input2,char* output,float min)	模板匹配，返回值是匹配框的左上角坐标（x，y）。min 是与匹配分数相关的参数，参考：min=65026。

int* TemplateMatching(bmp_in input1,bmp_in input2,float min)	模板匹配，返回值是匹配框的左上角坐标（x，y）。min 是与匹配分数相关的参数，参考：min=65026。
	需引入以下结构体：
	struct bmp_in {
	int num_components, rows, cols;
	int num_unread_rows;
	int line_bytes;
	int alignment_bytes;
	FILE *in;
	};
	声明：
	extern int bmp_in__open(bmp_in *state, const char *fname);
	参考：
	bmp_in input1,input2;
	bmp_in__open(&input1,input1file);

	bmp_in__open(&input2,input2file);

double* TemplateMatch(byte*** input,byte*** Template,char* output,int irows,int icols,int trows,int tcols,int size,int best_loss,double a,double b,double c,double d,int e1,int e2)	模板匹配，支持 JPG 图像，返回值的第 1 和第 2 个元素是匹配框左上角顶点的横坐标和纵坐标，第 3 个元素是目标与模板相对的旋转角度，第 4 个元素是缩放比例。参考：size=1，best_loss=1000000000，a=0.5，b=2.1，c=0.5，d=45，e1=20，e2=20。
	声明：
	#define byte unsigned char
	byte ***LoadRgb(const char *fname, int *rows, int *cols, int *chan);
	参考：
	int irows, icols, ichan;
	int trows, tcols, tchan;
	byte*** input = LoadRgb(inputfile, &irows, &icols, &ichan);
	byte*** Template = LoadRgb(templatefile, &trows, &tcols, &tchan);

double* TemplateMatch(byte*** input,byte*** Template,int irows,int icols,int trows,int tcols,int size,int best_loss,double a,double b,double c,double d,int e1,int e2)	模板匹配，支持 JPG 图像，返回值的第 1 和第 2 个元素是匹配框左上角顶点的横坐标和纵坐标，第 3 个元素是目标与模板相对的旋转角度，第 4 个元素是缩放比例。参考：size=1，best_loss=1000000000，a=0.5，b=2.1，c=0.5，d=45，e1=20，e2=20。
	声明：
	#define byte unsigned char
	byte ***LoadRgb(const char *fname, int *rows, int *cols, int *chan);
	参考：
	int irows, icols, ichan;
	int trows, tcols, tchan;
	byte*** input = LoadRgb(inputfile, &irows, &icols, &ichan);
	byte*** Template = LoadRgb(templatefile, &trows, &tcols, &tchan);

byte*** TemplateMatch1(byte*** input,byte*** Template,int irows,int icols,int trows,int tcols,int size,int best_loss,double a,double b,double c,double d,int e1,int e2)	模板匹配，支持 JPG 图像，返回匹配结果。参考：size=1，best_loss=1000000000，a=0.5，b=2.1，c=0.5，d=45，e1=20，e2=20。
	声明：
	#define byte unsigned char
	byte ***LoadRgb(const char *fname, int *rows, int *cols, int *chan);
	void SaveRgbPng(byte ***in, const char *fname, int rows, int cols);
	参考：
	int irows, icols, ichan;

	int trows, tcols, tchan;
	byte*** input = LoadRgb(inputfile, &irows, &icols, &ichan);
	byte*** Template = LoadRgb(templatefile, &trows, &tcols, &tchan);

int ObjectFind(bmpread_t input,bmpread_t Template)	模板匹配，返回匹配到的目标数量。
	需引入以下头文件：
	#include "bmpread1.h"
	参考：
	bmpread_t input, Template;
	bmpread(inputfile, BMPREAD_BYTE_ALIGN | BMPREAD_ANY_SIZE, &input);
	bmpread(Templatefile, BMPREAD_BYTE_ALIGN | BMPREAD_ANY_SIZE, &Template);

float* ImageMatching(char* TargetImage,char* Template0,char* Template1,char* Template2,char* Template3,char* Template4,char* Template5,char* Template6,char* Template7,char* Template8,char* Template9)	图像匹配，返回值中的前 10 个元素是按顺序对应的目标与模板的差异分数,最后一个元素是匹配到的模板的序号。
float* ImageMatching(char* TargetImage, char* Template0, char* Template1)	图像匹配,返回值中的前 2 个元素是按顺序对应的目标与模板的差异分数,最后一个元素是匹配到的模板的序号。支持 BMP 图像。
float* ImageMatching(Image3 TargetImage,float **templates,int num_templates)	图像匹配，如果模板数为 n，则返回值中的前 n 个元素是按顺序对应的目标与模板的差异分数,最后一个元素是匹配到的模板的序号。 需引入以下结构体： typedef struct { float *data; int width; int height; }Image3; 声明：

	Image3 load_bmp(char *filename);
	
	Image3 img,TargetImage;
	float **templates;
	int num_templates = 10;
	templates = (float **) malloc(sizeof(float *) * num_templates);
	img = load_bmp(Template0);
	templates[0] = img.data;
	img = load_bmp(Template1);
	templates[1] = img.data;
	img = load_bmp(Template2);
	templates[2] = img.data;
	img = load_bmp(Template3);
	templates[3] = img.data;
	img = load_bmp(Template4);
	templates[4] = img.data;
	img = load_bmp(Template5);
	templates[5] = img.data;
	img = load_bmp(Template6);
	templates[6] = img.data;
	img = load_bmp(Template7);
	templates[7] = img.data;
	img = load_bmp(Template8);
	templates[8] = img.data;
	img = load_bmp(Template9);
	templates[9] = img.data;
	TargetImage = load_bmp(TargetImagefile);

void ImageFeatures(char* input,char* kernel,char* output)	图像特征。 kernel 文件内容样例： 3 1 0 -1 0 -1 5 -1 0 -1 0 其中，3 表示尺寸为 3*3，1 表示内核的规模
void FileWrite(char* BMP,char* TXT)	图像隐写之文件写入，将文本文件写入图像。支持 32 位 BMP 图像。BMP 是要写入的图像文件名，TXT 是要写入图像的文本文件名。
void FileWriteOut(char* BMP,char* TXT)	图像隐写之文件写出，将文本文件从图像中取出来。支持 32 位 BMP 图像。BMP 是要写出的图像文件名，TXT 是写出图像后信息保存的文本文件名。
void LBP(char* input,char* output)	LBP 图像特征提取。支持 PNG 图像。

void Watershed2(char* input,char* inputMarqueurs,char* output,int r,unsigned char R,unsigned char G,unsigned char B)	图像分割之分水岭算法。inputMarqueurs 是输入图像的标记图像。R=230，G=0，B=0，r=1。支持 PNG 图像。
void EcrireImage1(char* input,char* output,uint32_t rayon)	图像分割。rayon=5。支持 PNG 图像。
void EcrireImage2(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	图像分割。rayon=5。支持 PNG 图像。
void EcrireLPECouleur1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	图像分割。rayon=5。支持 PNG 图像。
void Watershed1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	图像分割之分水岭算法。inputMarqueurs 是输入图像的标记图像。rayon=5。支持 PNG 图像。
void EcrireImage3(char* input,char* inputMarqueurs,char* output,uint16_t rayon)	图像分割。rayon=1。支持 PNG 图像。
void EcrireImageCouleursAleatoires(char* input,char* inputMarqueurs,char* output,uint8_t r,uint8_t g,uint8_t	图像分割。rayon=1。支持 PNG 图像。

b,uint16_t rayon)	
void Watershed(char* input,char* inputMarqueurs,char* output,uint8_t r,uint8_t g,uint8_t b,uint8_t a,uint16_t rayon)	图像分割。inputMarqueurs 是输入图像的标记图像。a 一般为 255，rayon=1。支持 PNG 图像。
void FloodFill(char* input,char* output,int x,int y,unsigned char novaCor)	图像分割之漫水填充法。x=0，y=0，novaCor=127。支持 PGM 文件。
void ConvertCoordinatesToGraphics(char* input,char* output, double IMAGE_SIZE, double PIXEL_PADDING, bool drawlines)	读取点坐标并输出点之间绘制的点或线段的图像。IMAGE_SIZE=800，PIXEL_PADDING=25，drawlines=0 或 drawlines=1。 输入文件格式 如果我们将“N”表示为点数，则假设采用以下点坐标文件格式: N 1 x 坐标 y 坐标 2 x 坐标 y 坐标 3 x 坐标 y 坐标 4 x 坐标 y 坐标 .... N x 坐标 y 坐标
void HumanDetection1(char* input,char* output, double MINH, double MAXH, double MINS, double MAXS)	人类检测。参考：MINH=0.0， MAXH=50.0，MINS=0.23，MAXS=0.68。支持 24 位 BMP 图像。
void HumanDetection2(char* input, char* output, double MINH, double MAXH, double MINS, double MAXS)	人类检测。参考：MINH=0.0， MAXH=50.0，MINS=0.23，MAXS=0.68。支持 24 位 BMP 图像。
void HumanDetection3(char* input, char* output, double MINH, double MAXH, double MINS, double MAXS)	人类检测。参考：MINH=0.0， MAXH=50.0，MINS=0.23，MAXS=0.68。支持 24 位 BMP 图像。

int ImageFeatureNumber(char* input)	计算图像的特征点数量。支持 24 位 BMP 图像。
int ContentSimilarity(char* input1, char* input2,int KDTREE_BBF_MAX_NN_CHKS,double NN_SQ_DIST_RATIO_THR)	返回两个图像的内容相似度。KDTREE_BBF_MAX_NN_CHKS 是在 BBF 搜索期间要检查的关键点 NN 候选的最大数量，NN_SQ_DIST_RATIO_THR 是 NN 和第二个 NN 之间距离平方比的阈值,参考:KDTREE_BBF_MAX_NN_CHKS=100，NN_SQ_DIST_RATIO_THR=0.49。支持 24 位 BMP 图像。
Feature* ImageFeature(char* input)	返回图像的特征数据。支持 24 位 BMP 图像。 需引入一下结构体： typedef struct Point2D64f { double x; double y; }Point2D64f; typedef struct feature { double x; /x 坐标/ double y; /y 坐标/ double a; /Oxford 型仿射区域参数/ double b; /Oxford 型仿射区域参数/ double c; /Oxford 型仿射区域参数/ double scl; /Lowe 风格特征的比例/ double ori; /Lowe 风格特征的方向/ int d; /描述符长度/ double descr[128]; /描述符/ int type; /功能类型，OXFD 或 LOWE/ int category; /通用功能类别/ struct feature* fwd_match; /前向图像的匹配特征/ struct feature* bck_match; /从 backmward 图像中匹配特征/

	struct feature* mdl_match; /匹配模型中的特征/ Point2D64f img_pt; /图像中的位置/ Point2D64f mdl_pt; /模型中的位置/ void* feature_data; /用户可定义数据/ }Feature;
double CharacterRecognition (char* TargetImage,char* TemplateFileGroup[])	字符匹配，支持 BMP 图像，返回值是目标图像匹配到的模板文件的序号,如返回值是 2 则说明图像与序号为 2（序号从零开始）的模板匹配。 参考：TemplateFileGroup[]={ "0.txt", "1.txt", "2.txt", "3.txt", "4.txt", "5.txt", "6.txt", "7.txt", "8.txt", "9.txt" };
double CharacterRecognition 1(char* TargetImage,char* TemplateFileGroup[])	字符匹配，支持 BMP 图像，返回值是目标图像匹配到的模板文件的序号,如返回值是 2 则说明图像与序号为 2（序号从零开始）的模板匹配。 参考：TemplateFileGroup[]={ "0.txt", "1.txt", "2.txt", "3.txt", "4.txt", "5.txt", "6.txt", "7.txt", "8.txt", "9.txt" };