User Manual

Catalogue

PPM, PGM, and PBM image processing

void OTSUBinarization(char* input,char* output)	OTSU binarization.input is the input file name, output is the output file name.PGM images in P5 format are supported.
void PPMtoBMP(char* input,char* output,int bpp)	Convert PPM images to BMP images. bpp is the color depth of a BMP image.
void BMPtoPPM(char* input,char* output)	Convert BMP images to PPM images.
void PPMtoBMP1(char* input,char* output,int bpp)	Convert PPM images to BMP images. bpp is the color depth of a BMP image.
void BMPtoPPM1(char* input,char* output)	Convert BMP images to PPM images.
void BMPtoPGM(char* input, char* output)	Convert BMP images to PGM images.
void BMPtoPPM2(char* input, char* output)	Convert BMP images to PPM images.
void PPMtoPGM(char* input, char* output)	Convert PPM images to PGM images.
void BlurPPM(char* input, char* output)	PPM image filtering.
void BlurPGM(char* input, char* output)	PGM image filtering.
void SegmentsOTSUBinarization(char* input,char* output)	OTSU binarization division.input is the input file name, output is the output file name.PGM images in P5 format are supported.

void P3PPMBlur(char* input,char* output)	PPM image Blur, input is the input file name, output is the output file name. Supports PPM images in P3 format.
unsigned char** ReadPBM(char* input)	Read the PBM image and return image data.input is the name of the PBM image file to read. Supports PBM images in P4 format.
void WritePBM(unsigned char** Input,char* output)	Save the PBM image.input is the input image data, and output is the output file name. Supports PBM images in P4 format.
void PGMHistogramEqualization(char* input,char* output)	Histogram equalization, input is the input file name, output is the output file name. Supports PGM images in P5 format.

PPMImage* ReadPPM(char* input)	PPM image reading, where input is the name of the PPM image file to be read. Support PPM images in P6 format.
	Structure to be introduced:
	typedef struct {
	unsigned char red, green, blue;
	// The color of pixels is represented by RGB (red/green/blue)
	} PPMPixel;
	typedef struct {
	unsigned int width, height;
	// The width and height of the image in pixels
	PPMPixel *data;
	// The pixels that make up the image
	} PPMImage;

void WritePPM(char* output,PPMImage* img)	Save PPM images, where output is the name of the output PPM image file and img is the input image data. Support PPM images in P6 format.
	Structure to be introduced:
	typedef struct {
	unsigned char red, green, blue;
	//The color of pixels is

	represented by RGB (red/green/blue)
	} PPMPixel;
	typedef struct {
	unsigned int width, height;
	//The width and height of the image in pixels
	PPMPixel *data;
	//The pixels that make up the image
	} PPMImage;

void InvertColor(char* input,char* output)	Negative filter, where input is the input file name and output is the output file name. Support PPM images in P6 format.
void GrayFilter(char* input,char* output)	Grayscale filter, where input is the input file name and output is the output file name. Support PPM images in P6 format.
void SepiaFilter(char* input,char* output)	Sepia ink filter, where input is the input file name and output is the output file name. Support PPM images in P6 format.
void AdjustSaturation(char* input,char* output,double a)	Adjust image saturation, where input is the input file name and output is the output file name. a is the target saturation, such as a=30. Support PPM images in P6 format.
void Resize(char* input,char* output,unsigned int NewWidth, unsigned int NewHeight)	Adjust the image size, where input is the input file name and output is the output file name. NewWidth and NewHeight are the width and height of the output image, respectively. Support PPM images in P6 format.
void AdjustHue(char* input,char* output,int a)	Adjust the color tone of the image, where input is the input file name and output is the output file name. a is the target color tone, such as a=125. Support PPM images in P6 format.
void AdjustBrightness(char* input,char* output,double a)	Adjust the brightness of the image, where input is the input

	file name and output is the output file name. a is the target brightness, such as a=60. Support PPM images in P6 format.
void AdjustContrast(char* input,char* output,double a)	Adjust the image contrast, where input is the input file name and output is the output file name. a is the target contrast, such as a=60. Support PPM images in P6 format.
void AdjustBlur(char* input,char* output,double a)	Blur the image using the sigma factor, where input is the input file name and output is the output file name. a is the sigma factor, such as a=5. Support PPM images in P6 format.
void MeanGrayFilter(char* input,char* output,double a)	Average grayscale filter, where input is the input file name and output is the output file name. a is the average coefficient, such as a=3. Support PPM images in P6 format.
void Pixelate(char* input,char* output,unsigned int a)	Pixarization, where input is the input file name and output is the output file name. a is the amplitude value, such as a=8. Support PPM images in P6 format.
void Rotate(char* input,char* output,short a)	Rotate the image, where input is the input file name and output is the output file name. a is the angle of rotation, such as a=45. Support PPM images in P6 format.
void GammaCorrection(char* input,char* output,double a)	Gamma correction, where input is the input file name and output is the output file name. a is the gamma number, such as a=0.5. Support PPM images in P6 format.
void GrayAndChannelSeparation(char* input,char* Grayoutput,char* Routput,char* Goutput,char* Boutput)	Generate grayscale images and RGB channel separation, with input being the input PPM image in P6 format; Grayoutput is the file name of the output grayscale image, while Routput, Goutput, and Boutput are the image file names

	of the output R, G, and B channels, respectively. The output is in PGM format.
void PGMBin(char* input,char* output,int threshold)	Grayscale image binarization, where the input is a grayscale image, the input and output are PGM files, and threshold is the threshold, such as threshold=125.
void Brightening(char* input,char* output,int a)	Color image enhancement, both input and output are PPM images in P6 format, where a is the enhancement coefficient, such as a=80.
void GrayBrightening(char* input,char* output,int a)	The grayscale image is brightened, and both the input and output are PGM images, where a is the brightening coefficient, such as a=80.
void PPMFilter(char* input,char* output)	Color image filtering, input and output are both P6 format PPM files.
void PGMGrayFilter(char* input,char* output)	Grayscale image filtering, both input and output are PGM images.
void PPMtoBMP(char* input,char* output)	Convert PPM images to BMP images, where input is the input file name and output is the output file name. Support PPM images in P6 format.
void PGMRotated(char* input,char* output,int width,int height,int channels,double theta)	Image rotation, channels are the channels for inputting images, theta is the rotation radian, supporting PGM images. Reference: theta=45.0*3.1415926/180。
void XCorner(char* input,char* output,int width,int height,int channels,double theta)	The angle in the X direction, channels, is the input channel for the image. Supports PGM images.
void YCorner(char* input,char* output,int width,int height,int channels,double theta)	The angle in the Y direction, channels, is the input channel for the image. Supports PGM images.
void Smooth(char* input,char* output,int width,int height,int channels,float sigma_x,float sigma_y,double theta)	Image blur, channels are the channels for input images, sigma_x is the ambiguity coefficient in the X direction, sigma_y is the blur coefficient in the Y direction.

void PGMOtsuThreshold(string input,char* output)	Otsu threshold method, where input is the input file name and output is the output file name. Supports PGM images in P5 format.
void PGMLocalisedOtsuThreshold(string input,char* output)	Local Otsu threshold, where input is the input file name and output is the output file name. Supports PGM images in P5 format.
void PGMSauvolaThreshold(string input,char* output,double a,double b,double c)	Sovola threshold, supporting PGM images in P5 format. The reference values for a,b and c are as follows: a=0.01, b=15, c=225.
void PGMThreshold(string input,char* output,int thresh)	Threshold method, where input is the input file name and output is the output file name. Supports PGM images in P5 format. thresh is the threshold, such as: thresh=5.
float Repair1(char* input,char* output,float var,float threshold,int nbLevels,float a)	For inpainting, var is the noise variance, threshold is the threshold, nbLevels is the number of levels to be processed, a=10. Return to ISNR.
float Repair2(char* input,char* output,float var,float threshold,int nbLevels,float a)	For inpainting, var is the noise variance, threshold is the threshold, nbLevels is the number of levels to be processed, a=10. Return to ISNR.
void LowPassFilterRepair1(char* input,char* output,int size_filter,float var,int nb_iterations,int nbLevels,float a,int b)	Low pass filter inpainting, a=10, b=6, nbLevels=3, size_ Filter is the size of the low-pass filter, var is the noise variance, nb_iterations is the iteration algebra of Landweber.
void LowPassFilterRepair2(char* input,char* output,int size_filter,float var,int nb_iterations,int nbLevels,float a,int b)	Low pass filter inpainting, a=10, b=6, nbLevels=3, size_ Filter is the size of the low-pass filter, var is the noise variance, nb_iterations is the iteration algebra of Landweber.
float LowPassFilterRepair3(char* input,char* output,int size_filter,float var,int nb_iterations,int nbLevels,int pas,float a,int b)	Low pass filter inpainting, a=10, b=6, nbLevels=3, pas=1, size_ Filter is the size of the low-pass filter, var is the noise variance, nb_iterations is the iteration

	algebra of Landweber. Return to ISNR.
void Repair1(char* input,char* output,int M,float a)	Inpainting, a=0.0, M is the number of decomposition layers, such as M=3.
void Repair2(char* input,char* output,int M,float a)	Inpainting, a=0.0, M is the number of decomposition layers, such as M=3.
void MakeNoise1(char* input,char* output,int size_filter)	Manufacturing noise, size_ Filter is the width of the low-pass filter.
void MakeNoise2(char* input,char* output,int nb_iterations,int pas)	Manufacturing noise, nb_iterations is Landweber's iteration algebra, pas=1.
void MakeNoise3(char* output,int height,int width,float var)	To create noise, height is the height of the output image, width is the width of the output image, and var is the noise variance.
void MakeNoise4(char* input,char* output,int nb_iterations,int pas)	Manufacturing noise, nb_iterations is Landweber's iteration algebra, pas=1.
void ImageReconstruction(char* input,char* output,int maxDepth,int threshold,int tx,int ty)	Image reconstruction, supporting PGM files. Reference: maxDepth=80, threshold=50, tx=0, ty=0。

YUV image processing

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 stacking, Y_BLACK 、 U_BLACK and V_BLACK is used to turn the black color in the original image into transparent, Reference: 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)	YUV444 stacking, Y_BLACK 、 U_BLACK and V_BLACK is used to turn the black color in the original image into transparent, Reference: Y_BLACK=16,U_BLACK=128, V_BLACK=128。
void YUVsuperposition(char* input1,char*	Yuv444p is directly

input2,char* output,int width,int height,unsigned char Y_BLACK,unsigned char U_BLACK,unsigned char V_BLACK)	stacked on Yuv420p without conversion, Y_BLACK 、 U_BLACK and V_BLACK is used to turn the black color in the original image into transparent, Reference: Y_BLACK=16，U_BLACK=128，V_BLACK=128。
void YUV444toYUV420(char* input,char* output,int height,int width)	YUV444 to YUV420, where height is the height of the input YUV444 file and width is the width of the input YUV444 file.
void YUV444toYUV420(char* input,char* output,int height,int width,int frames)	YUV444 to YUV420, where height and width are the height and width of the input file, and frames are the frame numbers for the operations in the input file.
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 goes to stacking on YUV420, Y_ BLACK、U_ BLACK and V_ BLACK is used to turn the black color in the original image into transparent, Reference: Y_BLACK=16，U_BLACK=128 ，V_BLACK=128。
void YUVEdgeProcessingY(char* input,char* output,int width,int height,double k)	YUV edge processing, where input is the input file name and output is the output file name. Width and height are the width and height of the input image. Reference: k=0.5。
void YUVEdgeProcessingU(char* input,char* output,int width,int height,double k)	YUV edge processing, where input is the input file name and output is the output file name. Width and height are the

	width and height of the input image. Reference: k=0.5。
void YUVEdgeProcessingV(char* input,char* output,int width,int height,double k)	YUV edge processing, where input is the input file name and output is the output file name. Width and height are the width and height of the input image. Reference: k=0.5。
void BMPLoadedIntoYUV(char* inputBMP,char* inputYUV,char* output,int YUVwidth,int YUVheight,int depth,bool mt)	YUV loads BMP, inputBMP is the input BMP image, inputYUV is the input YUV image, inputYUV acts as a container, YUVwidth and YUVheight are the width and height of the input YUV image. Reference : depth=12 , mt=true。
void YUVEdgeProcessingHorizontalDirection(char* input,char* output,int width,int height,double k)	YUV only handles horizontal edge processing, with input being the input file name and output being the output file name. Width and height are the width and height of the input image. Reference: k=0.7。
void YUVVieoEdgeProcessing(char* input,char* output,int width,int height,int frame,int max_frame)	YUV video file edge processing, where input is the input file name and output is the output file name. Width and height are the width and height of the input image, frame is the frame number to be processed, max_frame is the maximum frame number.
void YUVScale(char* input,char* output,int	Zoom the yuv420 image.

inputWidth,int inputHeight,int outputWidth,int outputHeight)	Reference : inputWidth=1280 , inputHeight=720 , outputWidth=128 , outputHeight=72。
void NoiseTreatment(char* input,char* output,int width,int height,int TWICEwidth,int TWICEheight)	YUV noise processing.
void NoiseTreatment(char* input,char* output,int width,int height,int frame,int max_frame)	YUV noise processing.

RAW image processing

unsigned char** RAWRead(char* input,int height,int width)	Read RAW images.
void RAWWrite(unsigned char** input,char* output,int height,int width)	Save the RAW image.
void MBVQ(char* input,char* output,int width,int height)	MBVQ effect, where input is the input file name and output is the output file name. Width and height are the width and height of the output image.
void RAWtoPPM_red(char* input,char* output,int width, int height,DebayerAlgorithm algo)	Extract the red channel after converting RAW to PPM. Reference : width=4096 , height=3072 , algo=NEARESTNEIGHBOUR 或 LINEAR。 Support RAW12 format. The following enumeration needs to be introduced: enum DebayerAlgorithm { NEARESTNEIGHBOUR, LINEAR };
void RAWtoPPM_green1(char* input,char* output,int width, int height,DebayerAlgorithm algo)	Extract green 1 channel after converting RAW to PPM. Reference : width=4096 , height=3072 , algo=NEARESTNEIGHBOUR 或 LINEAR。 Support RAW12 format. The following enumeration needs to be introduced: enum DebayerAlgorithm {

	NEARESTNEIGHBOUR, LINEAR };

void RAWtoPPM_green2(char* input,char* output,int width, int height,DebayerAlgorithm algo)	Extract green 2 channels after converting RAW to PPM. Reference : width=4096 , height=3072 , algo=NEARESTNEIGHBOUR 或 LINEAR。 Support RAW12 format.
	The following enumeration needs to be introduced:
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RAWtoPPM_blue(char* input,char* output,int width, int height,DebayerAlgorithm algo)	Extract the blue channel after converting RAW to PPM. Reference : width=4096 , height=3072 , algo=NEARESTNEIGHBOUR 或 LINEAR。 Support RAW12 format.
	The following enumeration needs to be introduced:
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RAWtoPPM(char* input,char* output,int width, int height,DebayerAlgorithm algo)	Convert RAW to PPM. Reference: width=4096 , height=3072 , algo=NEARESTNEIGHBOUR 或 LINEAR。 Support RAW12 format.
	The following enumeration needs to be introduced:
	enum DebayerAlgorithm {
	NEARESTNEIGHBOUR,
	LINEAR
	};

void RawPowerTransformation(char* input,char* output,int width,int height,int c,float v)	Power transformation, where input is the name of the input RAW image file, output is the name of the output RAW image file, width is the width of the input image, and height is the height of the input image. The

	default is c=1, v=0.6. Support RAW images.
void RAWAvgFilter(char* input,char* output,int ROWS,int COLS,int M,float mask[3][3])	Average filter, where input is the input file name and output is the output file name. ROWS is the row size of the image, COLS is the column size of the image, and M is the filtering related parameter, such as M=1; Mask is a filter template. Support RAW images. Reference template: 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)	Image inversion, where input is the name of the input RAW image file, output is the name of the output RAW image file, width is the width of the input image, and height is the height of the input image. Support RAW images.
void RawHistogramEqualization(char* input,char* output,int width,int height)	Histogram equalization: input is the input RAW image file name, output is the output RAW image file name, width is the width of the input image, and height is the height of the input image. Support RAW images.
void RAWHistogramEqualization(char* input,char* output,int width,int height)	RAW histogram equalization, width and height are the width and height of the input image.
void RAWMedianFilter(char* input,char* output,int ROWS,int COLS,int M,int sequence[9])	Median filtering, where input is the input file name and output is the output file name. ROWS is the row of the image, COLS is the column of the image, and M is the filtering related parameter, such as M=1. Support RAW images. Reference template:

	int sequence[9]={0,0,0,0,0,0,0,0,0};
void RawtoBmp1(char* input, char* output,unsigned long Width, unsigned long Height)	Convert RAW images to BMP images, where input is the input file name and output is the output file name. Width and Height are the width and height of the input file.
void RawToBmp(char* input,char* output,int imageWidth,int imageHigth)	Convert RAW images to BMP images, where input is the input file name and output is the output file name. Supports images with equal width and height.
void RGBtoHSI(char* input,char* output)	RGB color model is converted to HIS model, input is the input file name, and output is the output file name. Supports 24 bit BMP images.
void RAWLaplacialSharpeningFilter(char* input,char* output,int ROWS,int COLS,int M,float w,float mask[3][3])	Laplace sharpening filter, where input is the input file name and output is the output file name. ROWS is the row size of the image, COLS is the column size of the image, and M and w are filtering related parameters, such as M=1, w=1; Mask is a filter template. Support RAW images. Reference template: float mask[3][3] = {{0,1,0}, {1,-4,1}, {0,1,0}};
void RawLaplacianEnhancement(char* input1,char* output1,int width,int height)	Laplace operator enhancement, input1 is the input RAW image file name, output1 is the output RAW image file name, width is the width of the input image, and height is the height of the input image. Support RAW images.
void CyanGray(char* input,char* output,int width,int height)	Cyan grayscale image.
void MagentaGray(char* input,char* output,int width,int height)	Magenta grayscale image.

void YellowGray(char* input,char* output,int width,int height)	Yellow grayscale image.
void Transfer(char* input,char* output,int width,int height)	Transfer function.
void Homography(char* input1,char* input2,char* input3,char* output,int width,int height,int newwidth,int newheight)	Monography.
void MovieEffect(char* input,char* output,int width,int height)	Movie effects.
void FixedThresholdMethod(char* input,char* output,int width,int height)	Shake color processing, fixed threshold method.
void RandomThresholdMethod(char* input,char* output,int width,int height)	Shake color processing, random threshold method.
void DitherMatrixMethod(char* input,char* output,int width,int height,int N)	Dithering processing, dithering matrix method, default N=2.
void NormalizedLogBuffer1(char* input,char* output,int width,int height)	Logarithmic transformation, normalized logarithm.
void NormalizedLogBuffer2(char* input,char* output,int width,int height)	Logarithmic transformation, normalized logarithm.
void TernaryGrayLevel1(char* input,char* output,int width,int height)	Triple grayscale.
void TernaryGrayLevel2(char* input,char* output,int width,int height)	Triple grayscale.
void BestEdgeMap1(char* input,char* output,int width,int height)	Best edge map.
void BestEdgeMap2(char* input,char* output,int width,int height)	Best edge map.
void Skeletonize(char* input,char* output,int width,int height)	Skeletonization.
void SeparableDiffusion(char* input,char* output,int width,int height)	Separable diffusion.
void Denoising(char* input1,char* input2,char* output,int width,int	Remove noise.

height)	
void Luminosity(char* input,char* output,int width,int height)	Brightness adjustment.
void Average(char* input,char* output,int width,int height)	Averaging.
void MinMax(char* input,char* output,int width,int height)	Min and Max.
void Shrink(char* input,char* output,int width,int height)	Contraction.
void BilinearTransformation(char* input,char* output,int width,int height,int newwidth,int newheight)	Bilinear transformation.
void DitherMatrixMethod(char* input,char* output,int width,int height,int N)	Fourth level jitter, default N=2.
void Dewarped1(char* input,char* output,int width,int height,int Offset,double a,double b)	Dewaxing. a is to check whether the radius is<=a in the output image, and then twist it. Reference: 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])	Dewaxing. a is to check whether the radius is<=a in the output image, and then twist it. Reference: Offset=256, a=256.5, b=0.5。 Dewaxing specification: 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)	Texture segmentation, default K=6, N=100.
void TextureSegmentation2(char* input,char* output,int width,int height,int K,int N)	Texture segmentation, default K=6, N=100.
void TextureClassification(vector <string> filename,char* output,int width,int height,int K,int N,int a)	Texture classification, where a is the number of images to be classified. For example, if there are three image names in filename, a=3; Output is the classification result file, formatted as a text file in txt format; The default is K=4 and N=1000.
void ErrorDiffusion1(char* input,char* output,int width,int height)	Error diffusion.
void ErrorDiffusion2(char* input,char* output,int width,int height)	Error diffusion.
void ErrorDiffusion3(char* input,char* output,int width,int height)	Error diffusion.
void Thin(char* input,char* output,int width,int height)	Image refinement.
void OilPainting(char* input,char* output,int width,int height,int N)	Oil painting effect, default N=2.
void OilPainting1(char* input,char* output,int width,int height,int N)	Oil painting effect, default N=2.
void AverageFiltering(char* inputfile,char* outputfile,int width,int height)	3*3 Average filtering.
void GeometricMeanFiltering(char* inputfile,char* outputfile,int width,int height)	3*3 Geometric mean filtering.
void MedianFiltering(char* inputfile,char* outputfile,int width,int height)	Median filtering.

void FFT(char* input,char* output,int width,int height)	FFT function.
void LowPassOrHighPassFiltering(char* input,char* output,int width,int height,int LOW_PASS,int DEGREE)	Low pass or high pass filtering. LOW_PASS=1 is low-pass filtering, otherwise it is high-pass filtering, DEGREE is the degree of filtering, such as DEGREE=0.
void IFFT(char* input,char* output,int width,int height,int LOW_PASS,int DEGREE)	IFFT function. LOW_PASS=1 is low-pass filtering, otherwise it is high-pass filtering, DEGREE is the degree of filtering, such as DEGREE=0.
void BMPtoRAW(char* inputfile,char* outputfile)	Convert BMP images to RAW images. Supports 24 BMP images.
void BMPtoRAW1(char* input,char* output)	Convert BMP images to RAW images. Supports 24 BMP images.

BMP image processing

unsigned char** BMPRead8(char* input)	Read the pixels of an 8-bit BMP image.
void GenerateImage8(char* output,unsigned char** color)	Generate an 8-bit BMP image, where output is the name of the generated image file and color is the pixel data.

BMPMat** BMPRead(char* input)	Read the pixels of 24-bit and 32-bit BMP images.
	The following structure needs to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; // Alpha channel for 32-bit BMP images only
	}BMPMat;

unsigned int BMPHeight(char* input)	Read the height of the BMP image.

unsigned int BMPWidth(char* input)	Read the width of the BMP image.

void GenerateImage(char* output,BMPMat** color,unsigned short type)	Generate 24 bit and 32 bit BMP images. type is equal to the number of digits in the image, such as type=24.
	Reference case:
	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)	Histogram equalization, supporting 8-bit and 16 bit BMP. Input is the input file name, and output is the output file name.
void Resize(char* input,char* output,int Height,int Width)	Image scaling, supporting 8-bit and 16-bit BMPs. Input is the input file name, and output is the output file name. Height and Width are the height and width of the output image.
double MeanBrightness(char* input)	Calculate the average brightness of the image, supporting 8-bit and 16-bit BMPs. input is the input file name.
int IsBitMap(FILE *fp)	Determine if it is a bitmap.
int getWidth(FILE *fp)	Obtain the width of the image.
int getHeight(FILE *fp)	Obtain the height of the image.
unsigned short getBit(FILE *fp)	Obtain the number of bits per pixel.
unsigned int getOffSet(FILE *fp)	The starting position for obtaining data.
void BMPtoYUV(char*	Convert BMP images to YUV images,

input,char* output, char yuvmode)	where input is the input file name and output is the output file name. yuvmode is the three mode options for YUV files, with values of '0', '2', and '4', respectively 420,422,444
void BMPtoYUV420I(char* input,char* output)	Convert BMP images to YUV420 images, where input is the input file name and output is the output file name.
void BMPtoYUV420II(char* input,char* output)	Convert BMP images to YUV420 images, where input is the input file name and output is the output file name.
void DCMtoBMP(string input,char* output)	Convert DCM images to BMP images. Input is the input file name, and output is the output file name.
void Ins1977(char* input,char* output,int ratio)	Ins1977 filter, where input is the input file name and output is the output file name. Reference : ratio=100。
void LOMO(char* input,char* DarkAngleInput,char* output,int ratio)	LOMO filter, DarkAngleInput is the name of the dark corner template image. Reference: ratio=100。
void PNGGray(char* input,char* output)	Grayscale the image, where input is the input file name and output is the output file name.
void PNGSpotlight(char* input,char* output,int centerX,int centerY,double a,double b,double c,double d,double e)	Spotlight effect, where input is the input file name and output is the output file name. Focus coordinates (centerX, centerY), such as: centerX=400, centerY=180; a, b, c, d, e are related parameters, with default values of a=100, b=100, c=160, d=80, e=0.5.
void PNGIllinify(char* input,char* output)	Phantom effect, where input is the input file name and output is the output file name.
void PNGWaterMark(char* input1,char* input2,char* output)	The image must be watermarked, and the dimensions of input1 and input2 must be the same.
void Short(char* input,char* output,int a,int b,int c,double d,int depth)	Dwarfing effect. a=1,b=128,c=2,d=0.5, depth=24。 Supports 24 bit BMP images.
void Rise(char* input,char* output,int a,int b,double c,int d,int depth)	Increase special effects. a=1, b=128, c=0.5, d=2, depth=24。 Supports 24 bit BMP images.

void Short1(char* input,char* output,int a,int b,double c,double d,int depth)	Dwarfing effect. a=1, b=128, c=0.5, d=0.5, depth=24。 Supports 24 bit BMP images.
void Handstand(char* input,char* output,int a,int b,double c,int depth)	Inverted special effect. a=1, b=128, c=0.5, depth=24。 Supports 24 bit BMP images.
void Fat(char* input,char* output,int a,int b,double c,int depth)	Obesity specific effects. a=1, b=128, c=0.5, depth=24。 Supports 24 bit BMP images.
void HighFoot(char* input,char* output,int a,int b,int c,double d,int depth)	High foot effect. a=1, b=128, c=2, d=0.5, depth=24。 Supports 24 bit BMP images.
void CurvedCurve(char* input,char* output,int a,int b,int c,int d,double e,int depth)	Curved special effect. a=1,b=128,c=4, d=2, e=0.5, depth=24。 Supports 24 bit BMP images.
void Thin(char* input,char* output,int a,int b,double c,double d,int depth)	Refine special effects. a=1, b=128, c=0.5,d=0.5,depth=24。Supports 24 bit BMP images.
void Winding(char* input,char* output,int lim,int a,int b,int c,int d,double e,int depth)	Bending effect. lim=20,a=1,b=128,c=4, d=5, e=0.5, depth=24。 Supports 24 bit BMP images.

void CrossDenoising(unsigned char** input,unsigned char** output,double a)	The cross method removes isolated pixels.
	The following structures and declarations need to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; // Alpha channel for 32-bit BMP images only
	}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)	The cross method removes isolated pixels.
	The following structures and declarations need to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; // Alpha channel for 32-bit BMP images only
	}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)	The crossover method removes isolated pixels.
	The following structures and declarations need to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; // Alpha channel for 32-bit BMP images only
	}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)	The crossover method removes isolated pixels.
	The following structures and declarations need to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel

	components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; //Alpha channel for 32-bit BMP images only
	}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** output,double a)	The matrix method removes isolated pixels.
	The following structures and declarations need to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; //Alpha channel for 32-bit BMP images only
	}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)	The matrix method removes isolated pixels.
	The following structures and declarations need to be introduced:
	typedef struct {
	unsigned char B; //Blue channel components of 24-bit and 32-bit BMP images
	unsigned char G; //Green channel components of 24-bit and 32-bit BMP images
	unsigned char R; //Red channel component of 24-bit and 32-bit BMP images
	unsigned char A; // Alpha channel for 32-bit BMP images only
	}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)	Fusion of multi focus images, supporting 8-bit BMP images. 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)	Fusion of multi focus images, supporting 8-bit BMP images. 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)	Image fusion. Reference: a=3, b1=4, DX1=-68, DY1=-99, EPS=1, input1="Image fusion1.jpg" , input2="Image fusion2.jpg", MaskImage="Mask.png", output="output.jpg"。 int dx[] = {0,0,-1,1}; int dy[] = {-1,1,0,0};
void Screenshot3(HWND hWnd, LPCWSTR OutputImage)	Screenshot function. hWnd is the window handle to be screenshot, such as: GetDesktopWindow(); OutputImage is the screenshot name.
void Screenshot1(HWND hWnd, LPCWSTR OutputImage)	Screenshot function. hWnd is the window handle to be screenshot, such as: GetDesktopWindow(); OutputImage is the screenshot name.
void Screenshot2(HWND hWnd,LPCWSTR OutputImage)	Screenshot function. hWnd is the window handle to be screenshot, such as: GetDesktopWindow(); OutputImage is the screenshot name.
void Dark(char* input,char* output,int ratio)	Dimming filter. Reference: ratio=100。
void WaveFilter(char* input,char* output,int degree,int a)	Wave deformation special effect filter, degree is the degree of filter (wave distortion). Generate BMP images when a=0, JPG images when a=1, PNG images when a=2, and TGA images when a=3. Reference: degree=10.
void PinchFilter(char* input,char* output,int a)	Squeeze deformation special effect filter, generate BMP image when a=0, JPG image when a=1, PNG image when a=2, and TGA image when a=3.
void PinchFilter(char*	Squeeze deformation special effect

input,char* output,int cenX,int cenY,int a)	filter, generate BMP image when a=0, JPG image when a=1, PNG image when a=2, TGA image when a=3, cenX is the X coordinate of the deformation center point, and cenY is the Y coordinate of the deformation center point.
void SpherizeFilter(char* input,char* output,int a)	The spherical deformation special effect filter generates BMP images when a=0, JPG images when a=1, PNG images when a=2, and TGA images when a=3.
void SpherizeFilter(char* input,char* output,int cenX,int cenY,int a)	The spherical deformation special effect filter generates a BMP image when a=0, a JPG image when a=1, a PNG image when a=2, and a TGA image when a=3. cenX is the X coordinate of the deformation center point, and cenY is the Y coordinate of the deformation center point.
void SwirlFilter(char* input,char* output,int ratio,int a)	Rotate the deformation special effect filter, generate BMP image when a=0, JPG image when a=1, PNG image when a=2, TGA image when a=3, ratio=3.
void SwirlFilter(char* input,char* output,int cenX,int cenY,int ratio,int a)	Rotate the deformation special effect filter, generate BMP image when a=0, JPG image when a=1, PNG image when a=2, TGA image when a=3, ratio=3, cenX is the X coordinate of the deformation center point, and cenY is the Y coordinate of the deformation center point.
void ClosedOperation(char* input,char* output)	Closed operation, where input is the input file name and output is the output file name. Supports 4-bit BMP images.
void ColorTransfer(char* input1,char* input2,char* output)	Color transfer.
void GrayImage1(char* input,char* output)	Histogram equalization.
void ChannelHisteq(char* input, char* output)	Histogram equalization.
void HSVtoRGB(char* input, char* output)	HSV to RGB。

void HistogramEqualizationOnGrayImage(string input,char* output)	Histogram equalization.
CImg<unsigned int> HistogramEqualizationOnGrayImage2(string input)	Histogram equalization.
void HistEqualColorImageOneColorChannel(string input, char* output)	Histogram equalization.
CImg<unsigned int> HistEqualColorImageOneColorChannel1(string input)	Histogram equalization.
void HistEqualColorImageThreeColorChannels(string input, char* output)	Histogram equalization.
CImg<unsigned int> HistEqualColorImageThreeColorChannels(string input)	Histogram equalization.
void HistEqualColorImageHSISpace(string input, char* output)	HSI Space.
CImg<unsigned int> HistEqualColorImageHSISpace(string input)	HSI Space.
void ColorTransfer1(char* sourceImage,string targetImage, char* output)	Color transfer.
CImg<unsigned int> ColorTransfer2(string sourceImage, string targetImage)	Color transfer.
void BMPtoJPG(char* input,char* output,int a)	Convert BMP images to JPG images. Supports 24 bit BMP images, and the size must be a multiple of 8. a represents the degree of file compression. The larger the number, the smaller the compressed file volume, such as a=100.
void PartialColorRetention(char* input,char* output,int	Partial color retention filters. Reference: ratio=60.

ratio)	
void GrayImageConversion8(char* input,char* output)	Generate grayscale images that support 8-bit BMP images. Input is the input file name, and output is the output file name.
void Gray(char* input,char* output)	Grayscale image conversion, supporting 24 bit BMP images. Input is the input file name, and output is the output file name.
void GrayImageConversion(char* input,char* output)	Color image to grayscale image, where input is the color image to be processed and output is the name of the grayscale image generated after processing. Supports 24 bit BMP images.
void BinaryImageVerticalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	The binary image is vertically mirrored. Input is the pixel data of the input image, output is the pixel data of the output image, w is the width of the input image, and h is the height of the input image.
void GrayImageVerticalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	The grayscale image is vertically mirrored, where input is the pixel data of the input image, output is the pixel data of the output image, w is the width of the input image, and h is the height of the input image.
void ColorImageVerticalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	Color images are vertically mirrored, where input is the pixel data of the input image, output is the pixel data of the output image, w is the width of the input image, and h is the height of the input image.
void OTSU(char* input,char* output,int BeforeThreshold)	Otsu algorithm, where input is the input file name and output is the output file name. BeforeThreshold is the initial threshold, such as BeforeThreshold=10. Supports 8-bit BMP images.
void LowerBrightness(char* input,char* output,int a,int b)	Turn down the brightness, where input is the input file name and output is the output file name. Supports 24 bit BMP images. The reference values for a and b can be a=100 and b=0.

void HightBrightness(char* input,char* output,int a,int b)	Turn up the brightness, where input is the input file name and output is the output file name. Supports 24 bit BMP images. The reference values for a and b can be a=100 and b=0.
void IterativeThresholdSelection(char* input,char* output)	Iteration threshold selection, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void DitheringMethod(char* input,char* output)	Jitter method, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void LogTransformation(char* input,char* output,int constant)	Logarithmic transformation, where input is the input file name and output is the output file name. Supports 8-bit BMP images. constant is a related parameter, such as constant=15.
void LogarithmicTransformation(char* input,char* output)	Logarithmic transformation, where input is the input file name and output is the output file name. Supports BMP images.
void HistogramEqualization(char* input,char* output)	Histogram equalization,input is the name of the input file and output is the name of the output file. Supports BMP images.
void Binarization(char* input,char* output,int threshold)	Binary, where input is the input file name and output is the output file name. Supports 24 bit BMP images. Threshold is the threshold, such as threshold=128.
void Expansion(char* input,char* output,unsigned char mask[9],int c)	The binary image expands. Reference：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)	Binary image corrosion. Reference：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)	Open operation of binary image. Reference：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。
void ClosedOperation(char* input,char* output,unsigned	Closed operation of binary image. Reference：

char mask[9],int c)	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)	Contour extraction from binary image by open operation. Reference : mask[9]={0,255,0,255,255,255,0,255,0}, c=128。
void ExpansionOperationToContourExtraction(char* input,char* output,unsigned char mask[9],int c)	The contour of binary image is extracted by dilation operation. Reference : mask[9]={0,255,0,255,255,255,0,255,0}, c=128。
void CorrosionCalculationToContourExtraction(char* input,char* output,unsigned char mask[9],int c)	The contour of binary image is extracted by etching operation. Reference : mask[9]={0,255,0,255,255,255,0,255,0}, c=128。
void Glaw(char* input,char* output,int ratio)	Luminous filter. Reference: ratio=100.
void LowPassFilter(char* input,char* output)	Low pass filter, where input is the input file name and output is the output file name. Supports BMP images.
void HighPassFilter(char* input,char* output)	High pass filter, where input is the input file name and output is the output file name. Supports BMP images.
void Thinning(char* input,char* output)	Image refinement, where input is the input file name and output is the output file name. Supports BMP images.
void ThinningLine(char* input,char* output)	The image is refined and linearized, with input being the input file name and output being the output file name. Supports BMP images.
void Corrosion(char* input,char* output)	Corrosion, input is the input file name, and output is the output file name. Supports 4-bit BMP images.
void Corrosion1(char* input,char* output,int *TempBuf, int TempH, int TempW)	Corrosion, input is the input file name, and output is the output file name. Supports 24 bit BMP images. TempBuf is a corrosion template, and TempH and TempW are the height and width of TempBuf, respectively. For example, if TempH=4 and TempW=4, there is TempBuf[4][4].
void Expand(char*	Inflation, input is the input file

input,char* output,int *TempBuf, int TempH, int TempW)	name, and output is the output file name. Supports 24 bit BMP images. TempBuf is an expansion template, and TempH and TempW are the height and width of TempBuf, respectively. For example, if TempH=4 and TempW=4, there is TempBuf[4][4].
unsigned char** create2DImg(unsigned char* input, int w, int h)	The grayscale image pixels stored linearly are converted into 2D.
unsigned char getMaxPixelWhole(unsigned char **input,int x,int y,int w,int h,int *Kernal,int kernalW,int halfKernalW)	Take the maximum value of the specified area of the image (to determine if it exceeds the boundary).
unsigned char getMaxPixelCenter(unsigned char **input,int x,int y,int *Kernal,int kernalW,int halfKernalW)	Take the maximum value of the specified area of the image (without determining whether it exceeds the boundary).
unsigned char** imgDilate(unsigned char *input,int w,int h,int *Kernal,int kernalW,int halfKernalW)	Image inflation.
unsigned char getMinPixelWhole(unsigned char **input,int x,int y,int w,int h,int *Kernal,int kernalW,int halfKernalW)	Take the minimum value of the specified area of the image (to determine if it exceeds the boundary).
unsigned char getMinPixelCenter(unsigned char **input,int x,int y,int *Kernal,int kernalW,int halfKernalW)	Take the minimum value of the specified area of the image (without determining whether it exceeds the boundary).
unsigned char** imgErode(unsigned char *input,int w,int h,int *Kernal,int kernalW,int halfKernalW)	Image corrosion.
void Corrosion(unsigned char *input,unsigned char *output,int rows,int cols,int mat[5][5])	Binary corrosion.

void Expansion(unsigned char *input,unsigned char *output,int rows,int cols,int mat[5][5])	Binary expansion.
void GaussianBlurFilter(char* input,char* output)	Gaussian filter, supporting PNG files.
void GaussianFiltering(char* input,char* output)	Gaussian filter, input is the name of the input file, and output is the name of the output file. Supports 24 bit BMP images.
void LaplaceEnhancement(char* input,char* output)	Laplace enhancement, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void Residual(char* input,char* output)	Find residuals, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void SunlightFilter(char* input,char* output,int intensity,int radius,int x,int y)	Illumination special effect filter, intensity is the intensity of the light, such as intensity=255; Radius is the lighting range, such as radius=600; x and y are the positions of illumination, such as x=100, y=60.
void Compress(char* input,char* output)	Compression, supporting multiple files. Input is the file name to be compressed, and output is the compressed file name.
void Decompression(char* input, char* output)	Decompression, supporting multiple files. Input is the name of the file to be extracted, and output is the name of the extracted file.
void BlackWhite(char* input,char* output)	Black and white conversion, where input is the original image of the input and output is the black and white image of the output. Supports 24 bit BMP images.
void Underexposure(char* input,char* output)	Image underexposure, where input is the original input image and output is the underexposed output image. Supports 24 bit BMP images.
void Overexposure(char* input,char* output)	Image overexposure, where input is the original input image and output is the

	overexposed output image. Supports 24 bit BMP images.
void Nostalgia(char* input,char* Mask,char* output,int ratio)	Nostalgia filter, input and Mask are both input file names, Mask is the wrinkled image path, ratio=100.
void GammaTransform(char* input,char* output)	Gamma transformation, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void GrayScale(char* input,char* output)	Grayscale, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void GrayImageBinarization(char* input,char* output,int bit,int threshold)	Grayscale image binarization, bit is used to set the number of bits, such as bit=8; Threshold is the threshold, such as threshold=200. Supports 8-bit BMP images.
void GreyPesudoColor(char* input,char* output)	Pseudo colorization of grayscale images, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void CalculateCumulativeHistogramMap(char* input,char* outfile)	Calculate the cumulative histogram and map it, with input being the input file name and output being the output file name. Supports 24 bit BMP images.
void Translation(string input,char* output,int dx,int dy)	Image translation, where input is the input file, dx and dy are the horizontal and vertical movement distances (in pixels), and negative values indicate left/down movement; output is the file name of the result after the translation operation. Supports BMP images.
void Mirrored(string input,char* output,char axis)	Mirror transformation, where input is the input file, output is the file name of the result after the mirror operation, and axis is the direction of the mirror transformation (represented by X or Y). Supports BMP images.
void Sheared(string input,char* output,char axis,double Coef)	Miscutting transformation, where input is the input file, output is the file name of the result after the miscutting operation, axis and Coef

	are the direction of the miscutting transformation (represented by X or Y) and the miscutting coefficient, respectively. Negative values are offset left/down. Supports BMP images.
void Scaled(string input,char* output,double cx,double cy)	Scaling operation, where input is the input file, output is the result file name after the scaling operation, cx and cy are the horizontal and vertical scaling coefficients, respectively. A coefficient greater than 1 indicates stretching, and a coefficient less than 1 indicates compression. Supports BMP images.
void Rotated1(string input,char* output,double angle)	Image rotation, where input is the input file, output is the file name of the rotated image, and angle is the rotation angle in radians. Supports BMP images.
void SaltNoise(char* input,char* output,int a,int b,int c,int d)	Add salt and pepper noise, where a and b are noise related parameters, such as a=3 and b=3; C and d are color related parameters, such as c=0, d=255. Supports 8-bit BMP images.
void CrossProcess(char* input,char* output,int ratio)	Cross printing filter. Reference: ratio=100.
void Conversion8(unsigned char** input,short** output)	unsigned char** to short**, output is used to save the results (with the same size as input).
void Conversion8(short** input,unsigned char** output)	short** to unsigned char**, output is used to save the results (with the same size as input).
void Conversion8(unsigned char** input,int** output)	unsigned char** to int**, output is used to save the results (with the same size as input).
void Conversion8(int** input,unsigned char** output)	int** to unsigned char**, output is used to save the results (with the same size as input).
void Conversion8(unsigned char** input,unsigned int** output)	unsigned char** to unsigned int**, output is used to save the results (with the same size as input).
void Conversion8(unsigned int** input,unsigned char**	unsigned int** to unsigned char**, output is used to save the results

output)	(with the same size as input).
void Conversion8(unsigned char** input,float** output)	unsigned char** to float **, output is used to save the results (with the same size as input).
void Conversion8(float** input,unsigned char** output)	float ** to unsigned char**, output is used to save the results (with the same size as input).
void Conversion8(unsigned char** input,double** output)	unsigned char** to double **, output is used to save the results (with the same size as input).
void Conversion8(double** input,unsigned char** output)	double ** to unsigned char**, output is used to save the results (with the same size as input).
void Conversion8(unsigned char** input,char** output)	unsigned char** to char **, output is used to save the results (with the same size as input).
void Conversion8(char** input,unsigned char** output)	char ** to unsigned char**, output is used to save the results (with the same size as input).
void Conversion24(BMPMat** input,BMPMatshort** output)	BMPMat ** to BMPMatshort **, output is used to save the results (with the same size as input).
void Conversion24(BMPMatshort** input,BMPMat** output)	BMPMatshort ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion24(BMPMat** input,BMPMatint** output)	BMPMat ** to BMPMatint **, output is used to save the results (with the same size as input).
void Conversion24(BMPMatint** input,BMPMat** output)	BMPMatint ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion24(BMPMat** input,BMPMatfloat** output)	BMPMat ** to BMPMatfloat **, output is used to save the results (with the same size as input).
void Conversion24(BMPMatfloat** input,BMPMat** output)	BMPMatfloat ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion24(BMPMat** input,BMPMatdouble** output)	BMPMat ** to BMPMatdouble **, output is used to save the results (with the same size as input).
void Conversion24(BMPMatdouble** input,BMPMat** output)	BMPMatdouble ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion24(BMPMat**	BMPMat ** to BMPMatchar **, output is

input,BMPMatchar** output)	used to save the results (with the same size as input).
void Conversion24(BMPMatchar** input,BMPMat** output)	BMPMatchar ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion32(BMPMat** input,BMPMatshort** output)	BMPMat ** to BMPMatshort **, output is used to save the results (with the same size as input).
void Conversion32(BMPMatshort** input,BMPMat** output)	BMPMatshort ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion32(BMPMat** input,BMPMatint** output)	BMPMat ** to BMPMatint **, output is used to save the results (with the same size as input).
void Conversion32(BMPMatint** input,BMPMat** output)	BMPMatint ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion32(BMPMat** input,BMPMatfloat** output)	BMPMat ** to BMPMatfloat **, output is used to save the results (with the same size as input).
void Conversion32(BMPMatfloat** input,BMPMat** output)	BMPMatfloat ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion32(BMPMat** input,BMPMatdouble** output)	BMPMat ** to BMPMatdouble **, output is used to save the results (with the same size as input).
void Conversion32(BMPMatdouble** input,BMPMat** output)	BMPMatdouble ** to BMPMat **, output is used to save the results (with the same size as input).
void Conversion32(BMPMat** input,BMPMatchar** output)	BMPMat ** to BMPMatchar **, output is used to save the results (with the same size as input).
void Conversion32(BMPMatchar** input,BMPMat** output)	BMPMatchar ** to BMPMat **, output is used to save the results (with the same size as input).
void MeanFiltering(char* input,char* output)	Mean filtering, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void MeanFltering1(char* input,char* output)	Mean filtering, where input is the input file name and output is the output file name. Supports 8-bit and 24-bit BMP images.
void KapoorAlgorithm(char*	Kapoor algorithm, where input is the

input,char* output,int BeforeThreshold)	input file name and output is the output file name. BeforeThreshold is the initial threshold, such as BeforeThreshold=150. Supports 8-bit BMP images.
void OpenOperation(char* input,char* output)	Open operation, where input is the input file name and output is the output file name. Supports 4-bit BMP images.
void Diffusion(char* input,char* output,int ratio)	Diffusion filter. Reference: ratio=90.
void LapulasFiltering(char* readPath,char* writePath,float CoefArray[9],float coef)	Laplace filtering, readPath is the original image, and writePath is the file name of the processed image. Supports 8-bit BMP images. Reference values for each parameter: Definition * 3 Template (Laplace): float CoefArray[9]={1.0f,2.0f,1.0f,2.0f,4.0f,2.0f,1.0f,2.0f,1.0f}; Define the coefficient multiplied before the template (Laplace): float coef=(float)(1.0/16.0);
void ImageFiltering(char* input,char* output,float kernel[3][3])	Image filtering, where input is the input file name and output is the output file name. The kernel is a fuzzy kernel. Supports 24 bit BMP images.
void ComicStrip(char* input,char* output,int ratio)	Comics filter. Reference: ratio=100.
void BrightnessAdjustment1(char* input,char* output,int brightness,int contrast)	Brightness and contrast adjustment. Reference: brightness=-30, contrast=100.
void BrightnessAdjustment2(char* input,char* output,int brightness,int contrast)	Brightness and contrast adjustment. Reference: brightness=-30, contrast=100.
void ZeroFillingSymmetricExtension(char* input,char* output)	Zero padding and symmetric expansion, supporting 8-bit and 24-bit BMP images.
void PopArtStyle(char*	Pop art style filters. Reference:

input,char* output,int ratio)	ratio=100.
void LightLeakage(char* input,char* Mask,char* output,int ratio)	Leakage filter, input and Mask are both input image names, Mask is the leakage template image, ratio=90.
void LinearFiltering(char* input,char* output,short average[3][3])	Linear filtering, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Reference template: short average[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
void MedianFiltering(char* input,char* output,short average[3][3])	Median filtering, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Reference template: 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])	Sharpening filtering, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Reference template: 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])	Gradient sharpening, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Reference template: 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)	Arithmetic mean filter, input is the input file name, and output is the output file name. Supports 8-bit BMP images.
void GeometricMeanFilter(char* input,char* output)	For the geometric mean filter, input is the name of the input file and output is the name of the output file. Supports 8-bit BMP images.
void HarmonicMeanFilter(char* input,char* output)	Harmonic averaging filter, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void ContraHarmonicMeanFilter(char* input,char* output)	Anti harmonic averaging filter, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Filter(char* input,char* output)	Filter, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Mosaic(char* input,char* output,int x)	Mosaicize the image, where input is the input file name and output is the output file name. x is the size of the mosaic processed block. Supports 24 bit BMP images.
void MosaicFilter(char* input,char* output,int ratio)	Mosaic filter. Reference: ratio=50.
void Expansion(char* input,char* output)	Inflation, input is the input file name, and output is the output file name. Supports 4-bit BMP images.
void MidSmoothing(char* input,char* output)	Median filter: input is the name of the input file and output is the name of the output file. Supports 8-bit BMP images.
void AvgSmoothing(char* input,char* output)	Mean filter, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Averaging(char* input1,char* input2,char* input3,char* output,int a)	Image averaging, where input is the input file name and output is the output file name. a is the average related parameter, such as a=3. Supports 8-bit BMP images.

void PlaneSlicing(char* input,char* output)	Flat slice, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Translation(char* input,char* output,int xoffset,int yoffset)	Image translation, reference: xoffset=-100, yoffset=-100.
void SharpeningSpatialFiltering8(char* input,char* output,int model[9])	Sharpen spatial filter, where input is the input file name and output is the output file name. Model is a sharpened template. Supports 8-bit grayscale images.
void PseudoGrayscale(char* input,char* output)	Pseudo grayscale, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void TwoColors(char* input,char* output,int threshold,unsigned char color1,unsigned char color2)	Dichromization, where input is the input file name and output is the output file name. Threshold is the threshold, such as threshold=115; color1 and color2 are the two colors to fill. Supports 24 bit BMP images.
void PNGImageGeneration(char* filename,const unsigned char img[],unsigned W,unsigned H,int x)	Filename is the name of the generated PNG image file; img is the pixel data of the image, W is the width of the image, H is the height of the image, x=0 selects to generate an RGB image, and x=1 selects to generate an RGBA image.
void MakeSphere(double V[3],double S[3], double r, double a, double m, int ROWS, int COLS, char* output)	Using a reflection model to generate an image of a sphere under orthogonal projection, where V is the direction of the camera, output is the file name of the output result image, ROWS is the number of rows in the output image, and COLS is the number of columns in the output image. Reference：V[3] = {0.0, 0.0, 1.0}，S[3] = {0.0, 0.0, 1.0}， r=50， a=0.5， m=1.Support RAS files.
void MakeSphere(double vector_v[3],double vector_s[3], double r, double a, double m, int ROWS,int COLS,char* output,double	Generate an image of a sphere using a reflection model, vector_v is the direction of the camera, output is the file name of the output result image, ROWS is the number of rows in the

max)	output image, and COLS is the number of columns in the output image. Reference: 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.Support RAS files.
void BilateralFiltering(string input,char* output,double ssd, double sdid)	For Bilateral filter, input is the name of the input file and output is the name of the output file. Supports 24 bit BMP images. SSD and SDID are the standard deviations in the spatial domain and the standard deviations in the intensity domain, respectively.
void DoubleLayerErosion(char* input,char* output)	A double-layer morphological erosion with a circular structure set, supporting 8-bit and 24-bit BMP images.
void BinaryImageHorizontalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	Horizontal mirror image of binary image.
void GrayImageHorizontalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	Horizontal mirroring of grayscale images.
void ColorImageHorizontalMirror(unsigned char *input,unsigned char *output,unsigned int w,unsigned int h)	Horizontal mirroring of color images.
void SketchFilter(char* input,char* output,int ratio)	Sketch filter. Reference: ratio=100.
void Zoom(char* input,char* output,float scaleX,float scaleY,int interpolation)	Zoom. Reference: scaleX=5, scaleY=5, interpolation=0 or interpolation=1。
void AddGaussNoise(char* input,char* output)	Add Gaussian noise, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void AddSaltPepperNoise(char*	Add salt and pepper noise, where input is the input file name and output is

input,char* output)	the output file name. Supports 8-bit BMP images.
void ChannelSeparation(char* input,char* Routput,char* Goutput,char* Boutput)	Channel separation, where input is the input file name, Output is the red channel image, Gouutput is the green channel image, and Bouutput is the green channel image. Supports 24 bit BMP images.
void PatternMethod(char* input,char* output,unsigned char Template[8][8])	Pattern method, where input is the input file name and output is the output file name. Template is an array of templates. Supports 8-bit BMP images.
void LayerAlgorithm(char*input,char* inputMix,char* output,int alpha,int blendModel)	Layer algorithm, where input is the base layer image and inputMix is the mixed layer image. Reference : alpha=50, blendModel=26。 The corresponding pattern for the values of blendModel is as follows: 1 Typical 2 Dissolution 3 darkening 4 layers 5 Color Burn Mode 6 Linear deepening 7 tone 8 Brightening 9 Covering 10 color fade mode 11 Linear Dilution 12 light colors 13 stacking 14 Soft light mode 15 strong light mode 16 Bright mode 17 Linear light mode 18 point light mode 19 strong hybrid mode 20 differential 21 Exclusion mode 22 subtraction operation 23 Image segmentation 24 color mode 25 color saturation

	26 Coloring 27 brightness mode
void BMP24LossyCompression(char* input,char* output)	Image lossy compression, where input is the BMP file name to be compressed and output is the file name to be output after lossy compression. Supports 24 bit BMP images.
void BMP24LossyDecompression(char* input,char* output)	Image lossy decompression, where input is the file name to be decompressed and output is the BMP file name after decompression. Supports 24 bit BMP images.
void BMP24LosslessCompression(char* input,char* output)	Lossless image compression, where input is the BMP file name to be compressed and output is the file name output after lossless compression. Supports 24 bit BMP images.
void BMP24LosslessDecompression(char* input,char* output)	Lossless image decompression, where input is the file name to be decompressed and output is the BMP file name after decompression. Supports 24 bit BMP images.
void ImageDiscoloration(char* input,char* output,double a,double b,double c)	The image changes color, where input is the input file name and output is the output file name. For example, a=0.2126, b=0.7152, c=0.0722. Supports 24 bit BMP images.
unsigned char** HorizontalConcavity(unsigned char** input,int RANGE,int height,int width)	The horizontal concavity of image deformation returns the processing result. Reference: RANGE=400.
unsigned char** HorizontalConvexity(unsigned char** input,int RANGE,int height,int width)	The horizontal convexity of image deformation returns the processing result. Reference: RANGE=400.
unsigned char** TrapezoidalDeformation(unsigned char** input,int height,int width,double k)	The trapezoidal deformation of image deformation returns the processing result. Reference: k=0.3.
unsigned char** TriangularDeformation(unsigned char** input,int height,int width,double k)	Triangle deformation of image deformation, returns the processing result. Reference: k=0.5.
unsigned char**	S deformation of image deformation,

SDeformation(unsigned char** input,int height,int width,int RANGE)	returns the processing result. Reference: 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 linear detector. [in] src: Image, single channel [in] w: width [in] h: High [in] scaleX: The scaling factor on the X-axis [in] scaleY: The scaling factor on the Y-axis [in] bbox: The bounding box to be detected [in/out] lines: Results return: 0:ok; 1:error The following structures need to be introduced: 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)	Edge scoring detector. [in] src: Image, single channel [in] w: width [in] h: High [in] scaleX: The scaling factor on the X-axis [in] scaleY: The scaling factor on the Y-axis [in] bbox: The bounding box to be detected [in/out] lines: Results return: 0:ok; 1:error The following structures need to be

	introduced: 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)	Propagation filter. [in] src: Input image [in] guidance: guide image [in/out] dst: output image [in] w: width [in] h: High [in] c: Image channel, only c=1 or c=3 [in] r: Local window radius [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. return: 0:ok; 1:error
int PropagatedFilter2(unsigned char *src, unsigned char *guidance, unsigned char	Propagation filter. [in] src: Input image [in] guidance: guide image [in/out] dst: output image

*dst,int w, int h, int c, int r, float sigma_s, float sigma_r)	[in] w: width [in] h: High [in] c: Image channel, only c=1 or c=3 [in] r: Local window radius [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. return: 0:ok; 1:error
int BoxfilterFilter(unsigned char *src, unsigned char *dst,int w, int h, int c, int r)	Square box filtering. [in] src: Input image, single channel [in/out] dst: Output image, single channel [in] w: width [in] h: High [in] c: Image channel, only c=1 [in] r: Local window radius return: 0:ok; 1:error
int BoxfilterFilter1(unsigned char *src, unsigned char *dst,int w, int h, int c, int r)	Square box filtering. [in] src: Input image, single channel [in/out] dst: Output image, single channel [in] w: width [in] h: High [in] c: Image channel, only c=1 [in] r: Local window radius 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	Fast guided filtering [in] src: Input image, single channel [in] guidance: Guidance image, single channel [in/out] dst: Output image, single

r, float rp, float sr,float _scale)	channel [in] w: width [in] h: High [in] c: Image channel, only c=1 [in] r: Local window radius [in] rp: regularization Parameters: eps [in] sr: secondary sampling rate, sr>1: scale down, 0<sr<1: scale up If regularization _scale = 1; If not regularization _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)	Fast guided filtering [in] src: Input image, single channel [in] guidance: Guidance image, single channel [in/out] dst: Output image, single channel [in] w: width [in] h: High [in] c: Image channel, only c=1 [in] r: Local window radius [in] rp: regularization Parameters: eps [in] sr: secondary sampling rate, sr>1: scale down, 0<sr<1: scale up If regularization _scale = 1; If not regularization _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	Hoff line detector. [in] src: Image, single channel [in] w: width

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)	[in] h: High [in] scaleX: The scaling factor on the X-axis [in] scaleY: The scaling factor on the Y-axis [in] CannyLowThreshold: Low threshold for hysteresis processes in Canny operators [in] CannyHighThreshold: High threshold for hysteresis processes in Canny operators HoughRho: The distance resolution of the accumulator in pixels HoughTheta: The angle resolution of the accumulator in radians [in] MinThetaLinelength: Standard: For standard and multi-scale Hough transforms, check the minimum angle of the line Propagation ability: minimum line length. Line segments smaller than are rejected [in] MaxThetaGap: Standard: For standard and multi-scale Hough transforms, check the maximum angle of the line Probability based: maximum allowable gap between points connected to the same line HoughThreshold: Accumulator threshold parameter. Only those rows that receive sufficient votes will return (>threshold) [in] _type: hough Line method: hough_line_STANDARD or hough_line_PROBABILISTIC [in] bbox: The bounding box to be detected [in/out] lines: Results return 0:ok; 1:error _type: HOUGH_LINE_STANDARD: Standard Hough Line Algorithm HOUGH_LINE_PROBABILISTIC：Probability Hough Line Algorithm

	When HOUGH_LINE_STANDARD is running, the line points may be located outside of the image coordinates.
	
	standard: try (src,w,h,scalex,scaley,70,150, 1, PI/180, 0, PI, 100, HOUGH_LINE_STANDARD, bbox, line)
	Probabilistic: try (src,w,h,scalex,scaley,70,150, 1, PI/180, 30, 10, 80, HOUGH_LINE_STANDARD, bbox, line)。
	The following structures need to be introduced：
	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)	Fast Bilateral filter single channel. [in] src: Input image, single channel [in] guidance: Guidance image, single channel [in/out] dst: Output image, single channel [in] w: width [in] h: High

	[in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. If regularization _scale = 1; If not regularization _scale = 255*255 return: 0:ok; 1:error
int fast_bilateral_filter_single channel(unsigned char *src, unsigned char *guidance, unsigned char *dst, int w, int h, int c, float sigma_s, float sigma_r,float _scale)	Fast Bilateral filter single channel. [in] src: Input image, single channel [in] guidance: Guidance image, single channel [in/out] dst: Output image, single channel [in] w: width [in] h: High [in] c: Image channel, only c=1 [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region.

	If regularization _scale = 1; If not regularization _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)	Fast Bilateral filter RGB channel. [in] src: Input image, RGB channel [in/out] dst: Output image, RGB channel [in] w: width [in] h: High [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. If regularization _scale = 1; If not regularization _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)	Fast Bilateral filter RGB channel. [in] src: Input image, RGB channel [in/out] dst: Output image, RGB channel [in] w: width [in] h: High [in] c: Image channel, only c=3 [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace.

	[in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. If regularization _scale = 1; If not regularization _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)	Fast Bilateral filter. [in] src: Input image [in] guidance: Guide image, single channel, only a single channel is valid [in/out] dst: output image [in] w: width [in] h: High [in] c: Image channel, only c=1 or c=3 [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. If regularization _scale = 1; If not regularization _scale = 255*255 return: 0:ok; 1:error If the boot is NULL, the color filter can still be obtained
int permutohedral_bilateral_filter(unsigned char *src, unsigned char *guidance,	Fast Bilateral filter. [in] src: Input image [in] guidance: guide image [in/out] dst: output image

unsigned char *dst,int w, int h, int c, float sigma_s, float sigma_r,float _scale)	[in] w: width [in] h: High [in] c: Image channel, only c=1 or c=3 [in] sigma_s: Filter Sigma in Coordinate Space. The larger the value of the parameter, the more distant pixels will affect each other as long as the colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size without considering sigmaSpace. Otherwise, d is proportional to sigmaSpace. [in] sigma_r: Filter Sigma in Color Space. The larger the value of this parameter, the farther away colors within the pixel neighborhood (see sigmaSpace) will blend together, resulting in a larger semi isochromatic region. If regularization _scale = 1; If not regularization _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)	High pass filter. Reference : preserve=0。
void EmbossFilter(char* input,char* output,int preserve)	Relief filter. Reference: preserve=1。
void SharpenFilter(char* input,char* output,int preserve)	Sharpen the filter. Reference : preserve=1。
void Convolution(char* input,char* output,int w,int preserve)	Convolutional. Reference : w=7 , preserve=1。
void GaussianBlur(char* input,char* output,float sigma,int preserve)	Gaussian blur. Reference: sigma=2, preserve=1。
void HybridImage(char* input1,char* input2,char* output,float sigma,int preserve)	Blending images. Reference: sigma=2, preserve=1。
void LowFrequencyImage(char*	Low frequency images. Reference :

input,char* output,float sigma,int preserve)	sigma=2, preserve=1。
void HighFrequencyImage(char* input,char* output,float sigma,int preserve)	High frequency images. Reference：sigma=2, preserve=1。
void HighFrequencyImage1(char* input,char* output,float sigma,int preserve)	High frequency images. Reference：sigma=2, preserve=1。
void Bilateral(char* input,char* output,float sigma1, float sigma2)	Bilateral filter. Reference：sigma1=3, sigma2=0.1。
void SkinSmooth(char* input,char* output,int a,int b)	The skin is fine and smooth, a represents the smoothness level, b represents whether to apply skin filters, a=2, b=1.
void Resize1(char* input,char* output,int w, int h)	Image blur, w=713, h=467.
void Resize2(char* input,char* output,int w, int h)	Image blur.
void Shift(char* input,char* output,int ch, float v)	Shift function, ch=1, v=0.1.
void RGBtoHSV(char* input,char* output)	RGB to HSV.
void HSVtoRGB(char* input,char* output)	HSV to RGB.
void RGBtoLCH(char* input,char* output)	RGB to LCH.
void LCHtoRGB(char* input,char* output)	LCH to RGB.
void ColorTransfer(char* input1,char* input2,char* output)	Color transfer.
void DrawText(char* inputText,char* output,int width,int height,int depth,int spectrum,int x,int y, unsigned char R,unsigned char G,unsigned char B,unsigned char color1[],unsigned char	Text drawing, R=255, G=255, B=255, depth=1, spectrum=3, (x, y) is the coordinates of the text, color1 is the foreground color, color2 is the background color, opacity=1, font=60.

color2[],float opacity,unsigned int font)	
void EqualizedGray(char* input, char* output)	Histogram equalization of grayscale image.
void ColorHistogramEqualization(char* input, char* output)	Histogram equalization of color map.
void AverageHistogram(char* input, char* output)	Histogram equalization.
void HSIHist(char* input, char* output)	HIS histogram.
void ImageCutting(char* input,char* output,int leftdownx,int leftdowny,int rightupx,int rightupy)	Image cropping, where input is the input file name and output is the output file name. leftdownx, leftdowny, rightupx, rightupy are the coordinates of the bottom left and top right corners of the rectangular area to be cropped (four consecutive integer values, such as 50, 50, 300, 300). Supports 24 bit BMP images.
void ImageLayerAlgorithm(char* input,char* output)	Image layer algorithm.
void RGBtoGraywithoutLUT(char* input,char* output)	Grayscale image without LUT, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void RGBtoGraywithLUT(char* input,char* output)	The image has LUT grayscale, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void PiecewiseLinearTransform(char* input,char* output)	Piecewise linear transformation, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void PowerConvertion(char* input,char* output,double c,double g)	Power conversion, where input is the input file name and output is the output file name. For example, c=1.2, g=0.5. Supports 8-bit BMP images.
void Smooth(char* input,char* output)	Smooth, input is the input file name, and output is the output file name. Supports 8-bit BMP images.
void Multiply(char* input,char* output,int N,int	Image multiplication, where input is the input file name and output is the

SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3])	output file name. For example, N=1. Supports 8-bit BMP images.
	Reference template:
	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])	Image addition, where input is the input file name and output is the output file name. For example, N=1. Supports 8-bit BMP images.
	Reference template:
	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])	Power conversion, where input is the input file name and output is the output file name. For example, c=1.2, g=0.5, N=1. Supports 8-bit BMP images. 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)	Black-and-white image, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
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)	Feel free to operate, input is the input file name, and output is the output file name. Supports 24 bit BMP images.
void SpecialEffects1(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue)	Image special effects, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void GaussSmoothSharpen(char* input,char* output,int Template[3][3],int coefficient)	Gaussian smoothing, where input is the input file name and output is the output file name. Template is a Gaussian smoothing template with a coefficient of 16. Supports 24 bit BMP images.
void SmoothSharpen(char* input,char* output,int Template[3][3],int	Smooth, input is the input file name, and output is the output file name. Template is a smooth template,

coefficient)	homogenized, with coefficient1=9. Supports 24 bit BMP images.
static void NonmaximumWithoutDoubleThresholding(const LPCTSTR input, const LPCTSTR output,double a,double b,double c)	Reference: a=0.33, b=0.33, c=0.33. Supports 24 bit BMP images.
void AdjustPixel(char* input,char* output,int a)	Adjust pixel values, where input is the input file name and output is the output file name. A is used to set the relevant parameters of image pixels, such as a=3. Supports 24 bit BMP images.
void NostalgicFilter(BMPMat** input,BMPMat** output)	Retro filter, supporting 24 bit BMP images.
void SizeTransformation(short** input,short** output,short height,short width,short out_height,short out_width)	Image scaling, supporting 8-bit BMP images.
void ReverseColor(short** input,short** output,long height,long width,short GRAY_LEVELS)	Image inversion.
void Logarithm(short** input,short** output,long height,long width,short c)	Logarithmic transformation, default c=10.
void Gamma(short** input,short** output,long height,long width,double c)	Power law (gamma) transformation, default c=1.2.
void HistogramEqualization(short** input, short** output, long height, long width,short GRAY_LEVELS)	Histogram equalization.
void SmoothLinearFiltering(short** input, short** output,long height, long width,short average[3][3])	Smooth linear filter.
void MedianFiltering(short** input, short** output, long height, long width)	Median filter.

void Laplace(short** input,short** output,long height,long width,short sharpen[3][3])	Laplace operator.
void Sobel(short** input,short** output,long height,long width,short soble1[3][3],short soble2[3][3])	Sobel operator.
void DFTRead(short** input, double** output,long height,long width)	2D discrete Fourier transform, real part image.
void DFTImaginary(short** input,double** output,long height,long width)	2D discrete Fourier transform, imaginary part image.
void FreSpectrum(short **input, short **output,long height,long width)	Translation of Fourier transform.
void IDFT(double** re_array,double** im_array,short** output,long height,long width)	Two dimensional discrete Fourier inverse transform.
void AddGaussianNoise(short** input,short** output,long height,long width)	Add Gaussian noise.
void AddSaltPepperNoise(short** input, short** output,long height,long width)	Add salt and pepper noise.
void MeanFilter(short** input,short** output,long height,long width)	Mean filter.
void GeometricMeanFilter(short** input,short** output,long height,long width,double product)	Geometric mean filter, default product=1.0.
void HarmonicMeanFiltering(short** input,short** output,long height,long width,double sum)	Harmonic mean filtering, default sum=0.
void	Inverse harmonic mean filter, Q is the

InverseHarmonicMeanFiltering (short** input,short** output,long height,long width,int Q)	order of the filter, Q is positive to eliminate pepper noise, Q is negative to eliminate salt noise, Q=0 is arithmetic mean filter, Q=-1 harmonic mean filter, default Q=2.
void Threshold(short** input,short** output,long height,long width,int delt_t,double T)	Basic global threshold processing method.
void OTSU(short** input,short** output,long height,long width,short GRAY_LEVELS)	Otsu method for optimal global threshold processing.
void MatrixGlobalAddition24(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global addition based on template matrix.
void MatrixGlobalSubtraction24(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global subtraction based on template matrix.
void MatrixGlobalMultiplication24(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global multiplication based on template matrix.
void MatrixGlobalDivision24(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global division based on template matrix.
void MatrixGlobalAddition32(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global addition based on template matrix.
void MatrixGlobalSubtraction32(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global subtraction based on template matrix.
void MatrixGlobalMultiplication32(BMPMat** input1,BMPMat** input2,BMPMat** output)	Global multiplication based on template matrix.
void MatrixGlobalDivision32(BMPMat** input1,BMPMat**	Global division based on template matrix.

input2,BMPMat** output)	
void MatrixGlobalAddition8(unsigned char** input1,unsigned char** input2,unsigned char** output)	Global addition based on template matrix.
void MatrixGlobalSubtraction8(unsigned char** input1,unsigned char** input2,unsigned char** output)	Global subtraction based on template matrix.
void MatrixGlobalMultiplication8(unsigned char** input1,unsigned char** input2,unsigned char** output)	Global multiplication based on template matrix.
void MatrixGlobalDivision8(unsigned char** input1,unsigned char** input2,unsigned char** output)	Global division based on template matrix.

void ColorRectangleLocalSegmentation(char* input,char* output,int x1,int y1,int x2,int y2,BMPMat color)	The color image is partially truncated in a rectangular manner and filled with other parts. (x1, y1) is the coordinates of the upper left corner of the rectangle, and (x2, y2) is the coordinates of the lower right corner of the rectangle.
	Function source code:
	The following header file needs to be introduced:
	typedef struct {
	unsigned char B;
	unsigned char G;
	unsigned char R;
	unsigned char A;
	}BMPMat;
	State:
	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);
	Reference routine:
	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 GrayRectangleLocalSegmentation(char* input,char* output,int x1,int y1,int x2,int y2,unsigned char color)	The grayscale image is partially truncated in a rectangular manner and filled with other parts. (x1, y1) is the coordinates of the upper left corner of the rectangle, and (x2, y2) is the coordinates of the lower right

corner of the rectangle.
Function source code:
The following header file needs to be introduced:

```
typedef struct {
    unsigned char B;
    unsigned char G;
    unsigned char R;
    unsigned char A;
}BMPMat;
```

State:

```
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);
```

Reference routine:

```
    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)	Colorful drawing rectangle, (x1, y1) is the coordinates of the upper left corner of the rectangle, and (x2, y2) is the coordinates of the lower right corner of the rectangle.
	Function source code:
	The following header file needs to be introduced:
	typedef struct {
	unsigned char B;
	unsigned char G;
	unsigned char R;
	unsigned char A;
	}BMPMat;
	State:
	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);
	Reference routine:
	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)	Gray scale drawing rectangle, (x1, y1) is the coordinates of the upper left corner of the rectangle, and (x2, y2) is the coordinates of the lower right corner of the rectangle. Function source code: The following header file needs to be introduced: typedef struct {

	unsigned char B;
	unsigned char G;
	unsigned char R;
	unsigned char A;
	}BMPMat;
	State：
	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);
	Reference routine：
	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)	Relief effect, default value=128.
void Relief(unsigned char** input,unsigned char** output,int value)	Relief effect, default value=128.
void Sharpening(BMPMat** input,BMPMat** output,double degree)	Image sharpening, default degree=0.3.
void Sharpening(unsigned char** input,unsigned char** output,double degree)	Image sharpening, default degree=0.3.
void Soften(BMPMat** input,BMPMat** output,int value)	Image softening, default value=9.
void Soften(unsigned char** input,unsigned char** output,int value)	Image softening, default value=9.
void flipX(char* input,char* output)	Flip in X direction, supporting JPG files.
void flipY(char* input,char* output)	Flip in Y direction, supporting JPG files.
void Crop(char* input,char* output,uint16_t start_x, uint16_t start_y, uint16_t new_height, uint16_t new_width)	Cropping.
void Resize(char* input,char* output,int new_width, int new_height)	Zoom.
void Scale(char* input,char*	Proportion.

output,double ratio)	
void GrayscaleAvg(char* input,char* output)	Average grayscale value.
void grayscaleLum(char* input,char* output)	Grayscale brightness.
void ColorMask(char* input,char* output,float r,float g,float b)	Color mask.
void PixeLize(char* input,char* output,int strength)	Pixarization. Reference: length=2.
void GaussianBlur(char* input,char* output,int strength)	Gaussian blur. Reference: length=2.
void EdgeDetection(char* input,char* output,double cutoff)	Edge detection. Reference: cutoff=115.
void Sharpen(char* input,char* output)	Sharpening.
void CannyProcessing(char* input, char* output, int a)	Canny processing, a can be 1, 2, 3, 4, or 5. Supports BMP images.
void AverageGrayScale(char* input,char* output)	Average grayscale.
void SimpleBW(char* input,char* output)	Easy BW.
void AdvancedBW(char* input,char* output)	Advanced BW.
void UniformNoise(char* input,char* output)	Uniform noise.
void GaussianNoise(char* input,char* output,double sigma)	Gaussian noise.
void SaltAndPepperNoise(char* input,char* output)	Spicy salt noise.
void MeanFilter(char* input,char* output,int filterSize)	Mean filtering.
void GaussianFilter(char* input,char* output,double sigma)	Gaussian filter.
void MedianFilter(char* input,char* output,int size)	Median filtering.
void	Effective mean filter.

EfficientMeanFilter(char* input,char* output,int filterSize)	
double MeanSquaredError(char* input1,char* input2,char* output)	Mean square error, calculate image similarity, and the smaller the return value, the more similar the image will be.
void GrayAVS(char* input,char* output,float k,float b)	Input is the input file name, and output is the output file name. Supports 8-bit BMP images.
void HistogramEqualize24(char* input,char* output)	Histogram equalization: input is the name of the input file and output is the name of the output file. Supports 24 bit BMP images.
void MatrixTransformation(char* input,char* output)	Matrix transformation.
void Binarization(char* input,char* output)	Binarization.
void ChannelSeparation_B(char* input,char* output)	Separate the blue channel.
void ChannelSeparation_G(char* input,char* output)	Separate the green channel.
void ChannelSeparation_R(char* input,char* output)	Separate the red channel.
void Inverse(char* input,char* output)	Reversal.
void HistogramEqualization8(char* input,char* output)	Histogram equalization.
void Smooth(char* input,char* output)	Smooth.
void CannyEdge(char* input,char* output)	Canny operator.
void EdgeEnhance(char* input,char* output)	Edge enhancement.
void AvrFilter(char* input,char* output1,char* output2,int M,int N)	Input is the input file name, and output is the output file name. For example, M=21, N=1. Supports 8-bit BMP images.
void GryOppositionSSE(char*	Input is the input file name, and

input,char* output)	output is the output file name. Supports 8-bit BMP images.
void MedianFilter(char* input,char* output,int M,int N)	Median filter: input is the name of the input file and output is the name of the output file. For example, M=5, N=5. Supports 8-bit BMP images.
void EdgeSharpeningGry(char* input,char* output)	Input is the input file name, and output is the output file name. Supports 8-bit BMP images.
void SJGryandRiceTest(char* input,char* output)	Input is the input file name, and output is the output file name. Supports 8-bit BMP images.
void TextTest(char* input,char* output)	Input is the input file name, and output is the output file name. Supports 8-bit BMP images.
void RedChannel(char* input,char* output)	Generate a red channel image of the image, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void GreenChannel(char* input,char* output)	Generate a green channel image of the image, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void BlueChannel(char* input,char* output)	Generate a blue channel image of the image, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void HistogramStatistics(char* input,char* output)	Histogram statistics, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void HistogramEqualization1(char* input,char* output)	Histogram equalization: input is the name of the input file and output is the name of the output file. Supports 24 bit BMP images.
void ReflectionRay(char* input,char* output)	Reflection ray, input is the input file name, and output is the output file name. Supports 24 bit BMP images.
void MeanFiltering24(char* input,char* output)	Mean filtering, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void MedianFiltering24(char* input,char* output)	Median filtering, where input is the input file name and output is the

	output file name. Supports 24 bit BMP images.
void ZoomOutAndZoomIn(char* input,char* output,double value)	Scaling (bilinear interpolation), input is the input file name, and output is the output file name. value is the magnification, such as value=0.5. Supports 24 bit BMP images.
void Translation24(char* input,char* output,int x,int y)	Translation, where input is the input file name and output is the output file name. x is the translation of the horizontal axis, y is the translation of the vertical axis, such as x=-10, y=-30. Supports 24 bit BMP images.
void Mirror24(char* input,char* output)	Image, input is the input file name, and output is the output file name. Supports 24 bit BMP images.
void Rotate24(char* input,char* output,double degree)	Rotation, input is the input file name, and output is the output file name. Degree is the degree of rotation. Supports 24 bit BMP images.
void GivenThresholdMethod(char* input,char* output,int threshold)	Given the threshold method, the image is processed to black and white, with input being the input file name and output being the output file name. Threshold is the given threshold, such as threshold=100. Supports 24 bit BMP images.
void IterativeThresholdMethod(char* input,char* output)	The iterative threshold method processes images to make them black and white, with input being the input file name and output being the output file name. Supports 24 bit BMP images.
void OstuThresholdSegmentationMethod(char* input,char* output)	Ostu (Otsu method) threshold segmentation, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void Repudiation(char* input,char* output)	Reverse the pseudo color image, where input is the input file name and output is the output file name. Supports 24 bit BMP images.
void Gray1(char* input,char* output)	Convert a color image into a grayscale image, where input is the input file name and output is the output file name. Supports 24 bit BMP images.

void CorrectMethod(char* input,char* output)	The correct method is that input is the input file name and output is the output file name. Supports 24 bit BMP images.
void ChannelSeparation1(char* input,char* Routput,char* Goutput,char* Boutput)	Sort out the RGB components of the image and save them as independent images. input is the input file name, Routput is the red channel image, Gouutput is the green channel image, and Bouutput is the green channel image. Supports 24 bit BMP images.
void ReverseColor(char* input,char* output)	Invert the grayscale image, where input is the input file name and output is the output file name. Supports 8-bit BMP images.

Image1* LoadImage1(char* input)	BMP image reading, where input is the input file name. Supports 8-bit and 24-bit BMP images.
	Returns Image1 data, which has the following structure:
	typedef struct
	{
	int width;
	int height;
	int channels; //Number of image channels
	unsigned char* Data; //pixel data
	}Image1;

void SaveImage1(char* output,Image1* img)	Save Image1 data as a BMP image, where output is the name of the generated BMP image file and img is the image data to be saved. Supports 8-bit and 24-bit BMP images.
	The structure of Image1 data is as follows:
	typedef struct
	{
	int width;
	int height;
	int channels; // Number of image channels
	unsigned char* Data; // pixel data
	}Image1;

void ImageContrastExtension(char* input,char* output,double m,double g1,double g2,double a)	Image contrast extension, where input is the input file name and output is the output file name. Among them, reference can be made to: double m=1.5,g1=100.0,g2=200.0; m corresponds to the slope double a=(255.0-m*(g2-g1))/(255.0-(g2-g1)); Supports 8-bit BMP images.
void Binaryzation(char* input,char* output,int threshold)	Image binarization, where input is the input file name and output is the output file name. Threshold is the threshold for converting grayscale values into binary values, such as threshold=80. Supports 24 bit BMP images.
void GlobalBinarization(char* input,char* output)	Global binarization, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void AdaptiveBinarization(char* input,char* output)	Adaptive binarization, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void ExpansionOperation(char* input,char* output)	Expansion operation, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void CorrosionOperation(char* input,char* output)	Corrosion operation, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Operation1(char* input,char* output)	Open the operation, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Closed1(char* input,char* output)	Closed operation, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Negative1(char* input,char* output)	Image inversion, where input is the input file name and output is the output file name. Supports 24 bit BMP images.

void Negative(char* input,char* output)	Image inversion, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void ImageSynthesis(char* input1,char* input2,char* output)	Image synthesis.
void BlackWhite(char* input,char* output,float T,int border)	Black and white, supporting 8-bit and 24-bit BMP images. T is the threshold and border is the boundary range, such as T=50 and border=0.
void Mosaic(char* input,char* output,int w,int h)	Mosaicized images, where w and h are the width and height of the output image. Supports PNG images.
IMAGE Image_bmp_load(char* filename)	Load BMP images.
void Image_bmp_save(char* filename,IMAGE im)	Save BMP image.
IMAGE TransformShapeNearest(IMAGE input, unsigned int newWidth, unsigned int newHeight)	Zoom the image (nearest neighbor interpolation).
IMAGE TransformShapeLinear(IMAGE input, unsigned int newWidth, unsigned int newHeight)	Scale the picture (bilinear interpolation).
IMAGE TransformShapeWhirl(IMAGE input, float angle)	The rotation of an image at any angle.
IMAGE TransformShapeUpturn(IMAGE input, int a)	Mirror flipping of images.
void TransformColorGrayscale(IMAGE im, int GrayscaleMode)	Color image to grayscale image, for the values of GrayscaleMode: 1 represents weighted method, 2 represents maximum method, 3 represents mean method, 4 represents red component method, 5 represents green component method, and 6 represents blue component method.
void TransformColorBWDIY(IMAGE input, unsigned char Threshold)	Binary plot (custom threshold method).

void TransformColorBWOSTU(IMAGE input)	Binary plot (Otsu method OSTU, applicable to bimodal histogram.)
void TransformColorBWTRIANGLE(IMAGE input)	Binary plot (trigonometric TRIANGLE, applicable to unimodal histograms.)
IMAGE TransformColorBWAdaptive(IMAGE input, int areaSize)	Binary plot (adaptive threshold method, areaSize=25 is more suitable)
IMAGE TransformColorBWGrayscale(IMAGE input, int areaSize)	Binary map (using a binary map to represent grayscale changes, areaSize=25 is more appropriate)
void TransformColorOpposite(IMAGE input)	Reverse color.
IMAGE TransformColorHistogramPart(IMAGE input)	Histogram equalization (calculated step by step, the effect is softer).
IMAGE TransformColorHistogramAll(IMAGE input)	Histogram equalization (overall calculation, more sharp effect).
IMAGE KernelsUseDIY(IMAGE input, double* kernels, int areaSize, double modulus)	Convolutional operation (custom).
IMAGE WavefilteringMedian(IMAGE input)	Median filtering.

IMAGE WavefilteringGauss(IMAGE input,double KERNELS_Wave_Gauss[9],int a,double b)	Gaussian filter.
	Gaussian filter convolution kernel:
	double KERNELS_Wave_Gauss[9] =
	{
	1, 2, 1,
	2, 4, 2,
	1, 2 ,1
	};

IMAGE Wavefiltering_LowPass(IMAGE input, double* kernels)	Low pass filtering.
	// Low pass filtering convolutional kernel 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
	};

	// Low pass filtering convolutional kernel 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
	};
	
	// Low pass filtering convolutional kernel 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)	High pass filtering.
	//High pass filtering convolutional kernel HP1
	double KERNELS_Wave_HighPass_HP1[9] =
	{
	-1, -1, -1,
	-1, 9, -1,
	-1, -1 ,-1
	};
	
	// High pass filtering convolutional kernel HP2
	double KERNELS_Wave_HighPass_HP2[9] =
	{
	0, -1, 0,
	-1, 5, -1,
	0, -1 ,0
	};
	
	//High pass filtering convolutional kernel 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)	Mean filtering.
	//Mean filtering convolutional kernel
	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)	Differential edge detection.
	//Differential Vertical Edge Detection Convolutional Kernel
	double KERNELS_Edge_difference_vertical[9] =
	{
	0, 0, 0,
	-1, 1, 0,
	0, 0, 0
	};
	
	//Differential Horizontal Edge Detection Convolutional Kernel
	double KERNELS_Edge_difference_horizontal[9] =
	{
	0,-1, 0,
	0, 1, 0,
	0, 0, 0
	};
	
	//Differential Vertical and Horizontal Edge Detection Convolutional Kernel
	double KERNELS_Edge_difference_VH[9] =
	{
	-1, 0, 0,
	0, 1, 0,
	0, 0, 0
	};

IMAGE	Sobel edge detection.

KernelsUseEdgeSobel(IMAGE input, double* kernels1, double* kernels2)	//Sobel X edge detection convolutional kernel
	double KERNELS_Edge_Sobel_X[9] =
	{
	-1, 0, 1,
	- 2, 0, 2,
	-1, 0, 1
	};
	
	//Sobel Y edge detection convolutional kernel
	double KERNELS_Edge_Sobel_Y[9] =
	{
	-1, -2, -1,
	0, 0, 0,
	1, 2, 1
	};

IMAGE EdgeDetectionLaplace(IMAGE input, double* kernels)	Laplace edge detection.
	//Laplace edge detection convolutional kernel LAP1
	double KERNELS_Edge_Laplace_LAP1[9] =
	{
	0, 1, 0,
	1, -4, 1,
	0, 1, 0
	};
	
	//Laplace edge detection convolutional kernel LAP2
	double KERNELS_Edge_Laplace_LAP2[9] =
	{
	-1, -1, -1,
	-1, 8, -1,
	-1, -1, -1
	};
	
	//Laplace edge detection convolutional kernel LAP3
	double KERNELS_Edge_Laplace_LAP3[9] =
	{
	-1, -1, -1,
	-1, 9, -1,
	-1, -1, -1
	};

	//Laplace edge detection convolutional kernel LAP4
	double KERNELS_Edge_Laplace_LAP4[9] =
	{
	1, -2, 1,
	-2, 8, -2,
	1, -2, 1
	};

IMAGE MorphologyErosion(IMAGE input, double* kernels)	Corrosion.
	//Corrosive Convolutional Kernel
	double KERNELS_Morphology_Erosion_cross[9] =
	{
	0, 1, 0,
	1, 1, 1,
	0, 1, 0
	};

IMAGE MorphologyDilation(IMAGE input, double* kernels)	Expansion.
	//Expansive Convolutional Kernel
	double KERNELS_Morphology_Dilation_cross[9] =
	{
	0, 1, 0,
	1, 1, 1,
	0, 1, 0
	};

IMAGE Pooling(IMAGE input, int lenght)	Pooling.
IGIMAGE IntegralImage(IMAGE input)	Obtain the points chart (before this, make sure the picture is "black on a white background").
void FaceDetection(char* input,char* output,double* KERNELS_Wave_Average)	Face detection.
IMAGE FaceDetection(IMAGE input1,IMAGE input2,double* KERNELS_Wave_Average)	Face detection. The following structures need to be introduced: 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;
	State:
	IMAGE Image_bmp_load(char* filename);
	void Image_bmp_save(char* filename,IMAGE im);
	Reference:
	//For processing
	IMAGE input2 = Image_bmp_load(inputfile);
	//For saving
	IMAGE input2= Image_bmp_load(inputfile);
	input2=FaceDetection(input1,input2,KERNELS_Wave_Average);
	//Save Picture
	Image_bmp_save(outputfile, input2);

void IntegralDiagram(unsigned int *input, unsigned int *output, int width, int height)	Image integration chart.
void ImageEncryption(char* inFileName,char* outFileName,char key)	Image encryption, supporting 8-bit, 24-bit, and 32-bit BMP images. InFileName is the original image file name, outFileName is the decrypted image file name, and key is the key, such as key=255.
void ImageDecryption(char* inFileName,char* outFileName,char key)	Image decryption, inFileName is the encrypted image file name, outFileName is the decrypted image file name, and key is the key, such as key=255. Supports 8-bit, 24-bit, and 32-bit BMP images.
void Compress8(string input,string output)	Image compression, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void Decompression(string	Image decompression, where input is

input,string output)	the input file name and output is the output file name. Support the compressed result file of 8-bit BMP images.
void HorizontalMirror(char* input,char* output)	Horizontal mirroring, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void MirrorVertically(char* input,char* output)	Vertical mirroring, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void XMirroring(char* input,char* output)	X image, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void YMirroring(char* input,char* output)	Y image, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void ImageConvolution(char* input,char* output,double** Kernel,int n,int m)	Image convolution, where input is the input file name and output is the output file name. Kernel is a convolutional kernel, such as double Kernel [3] [3]={{-0.225, -0.225-0.225}, {-0.225,1, -0.225}, {-0.225, -0.225, -0.225}; n is the size of the first dimension of Kernel, and m is the size of the second dimension of Kernel, shaped like Kernel [n] [m]. Supports 24 bit BMP images.
void SpatialMeanFiter(char* input,char* output,int radius)	Spatial mean filter. Reference: radius=3.
void SpatialMedianFiter(char* input,char* output,int radius)	Spatial median filter. Reference: radius=3.
void SpatialMaxFiter(char* input,char* output,int radius)	Maximum space filter. Reference: radius=3.
void SpatialMinFiter(char* input,char* output,int radius)	Minimum space filter. Reference: radius=3.
void SpatialGaussFiter(char* input,char* output,int	Spatial Gaussian filter. Reference: radius=3.

radius)	
void SpatialStatisticalFiter(char* input,char* output,int radius,float T)	Spatial statistical filters. Reference: radius=3, T=0.2.
void FFTAmp(char* input,char* output,bool inv)	FFT amplifier. Reference: inv=false.
void FFTPhase(char* input,char* output,bool inv)	FFT phase. Reference: inv=false.
void STDFT1(char* input,char* output,bool inv)	Reference: inv=false.
void STDFT2(char* input,char* output,bool inv)	Reference: inv=false.
void SpectrumShaping(char* input,char* inputMsk,char* output)	Image frequency domain filtering, FFT transformation - phase spectrum, inputMsk is the name of the input mask image.
void Translation(char* input,char* output,int x,int y,unsigned char color)	Image translation, where input is the input file name and output is the output file name. X and y are the amount of translation on the X and Y axes, with the right as the positive direction, and color is the color filled in the non original image area after translation, such as color=100. Supports 8-bit BMP images.
void CrossDenoising24(BMPMat** input,BMPMat** output,BMPMat threshold,BMPMat target)	The image removes certain pixels, and the output is used to save the results (the same size as the input).
void CrossDenoising8(unsigned char** input,unsigned char** output,unsigned char threshold,unsigned char target)	The image removes certain pixels, and the output is used to save the results (the same size as the input).
void ImageDecontamination(BMPMat** input,BMPMat** output,int x1,int y1,int x2,int y2)	Image decontamination. (x1, y1) is the upper left corner coordinate of the rectangular stain area, and (x2, y2) is the lower right corner coordinate of the rectangular stain area.
void ImageDecontamination(unsigned char** input,unsigned	Image decontamination. (x1, y1) is the upper left corner coordinate of the rectangular stain area, and (x2, y2)

char** output,int x1,int y1,int x2,int y2)	is the lower right corner coordinate of the rectangular stain area.
void ImageSharpening(char* input,char* output)	Image sharpening, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void SharpenLaplace(char* input,char* output,int ratio)	Laplace sharpening. Reference: ratio=100.
void SharpenUSM(char* input,char* output,int radius,int amount,int threshold)	USM sharpening. Reference: 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)	Draw a rectangle on a 24 bit BMP image using the passed in parameters. Input is the input file name, and output is the output file name. (x1, y1) is the coordinates of the vertex on which the rectangle sits, and (x2, y2) is the coordinates of the lower right vertex of the rectangle; red is the red component of the rectangular wireframe, green is the green component of the rectangular wireframe, and blue is the blue component of the rectangle.
void GenerateBmp(unsigned char* pData,int width,int height,char* filename)	Generate a BMP image, where pData is the pixel data of the image, width and height are the width and height of the image, and filename is the file name of the generated image.
void Jpg24ImageGeneration(char* filename,unsigned int width, unsigned int height, unsigned char* img)	JPG image generation, where filename is the name of the generated JPG image file, width is the width of the image, height is the height of the image, and img is the pixel data of the image.
void ImageScalingNearestNeighborInterpolation(char* input,char* output,float lx,float ly)	The nearest neighbor interpolation method is used to remove the grid, where input is the input file name and output is the output file name. lx and ly are the multiples of length and width that need to be scaled. Supports 8-bit BMP images.
void	The bilinear interpolation method is

ImageScalingBilinearInterpolation(char* input,char* output,float lx,float ly)	used to remove the grid. Input is the name of the input file and output is the name of the output file. lx and ly are the multiples of length and width that need to be scaled. Supports 8-bit BMP images.
void BilinearInterpolationScaling(char* input,char* output,float ExpScalValue)	Bilinear interpolation, input is the input file name, and output is the output file name. ExpScalValue is the expected scaling factor (allowing decimals). Supports BMP images.
void NearestNeighborInterpolationScaling(char* input,char* output,float ExpScalValue)	Nearest neighbor interpolation, where input is the input file name and output is the output file name. ExpScalValue is the expected scaling factor (allowing decimals). Supports BMP images.
void ZoomImg(unsigned char *input,unsigned char *output,int sw,int sh,int channels,int dw,int dh)	Quadratic linear interpolation image scaling.
void CrossDenoising24(BMPMat** input,BMPMat** output,BMPMat target,BMPMatdouble weight)	Inpainting, output is used to save the results (the same size as input), target is the stain pixel, and weight is the repair weight coefficient.
void CrossDenoising8(unsigned char** input,unsigned char** output,unsigned char target,double weight)	Inpainting, output is used to save the results (the same size as input), target is the stain pixel, and weight is the repair weight coefficient.
void RotateRight90Degrees(char* input,char* output)	input is the input file name, and output is the output file name. Supports 8-bit BMP images, rotated 90 degrees to the right.
void RotateLeft90Degrees(char* input,char* output)	input is the input file name, and output is the output file name. Supports 8-bit BMP images, rotated 90 degrees to the left.
void ImageRotation(char* input,char* output,double angle)	Image rotation, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Angle is the angle to rotate.
void Rotation8(char* input,char* output,double	Image rotation, where input is the input file name and output is the

Angle,int x1,int y1,int x2,int y2,unsigned char color)	output file name. Supports 8-bit BMP images. Angle is the number of angles to rotate; x1, y1, x2, y2 are the coordinates of the center point around which the rotation revolves, and color is the fill color of the non original image area after rotation.
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)	Image rotation, where input is the input file name and output is the output file name. Supports 24 bit BMP images. Angle is the number of angles to rotate; x1, y1, x2, y2 are the coordinates of the center point around which the rotation revolves; Red, green, and blue are the red, green, and blue components of the colors to be filled in the non original image area after rotation.
void Rotation(char* input,char* output,int angle,unsigned char color)	Image rotation, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Angle is the angle of rotation, and color is the color used to fill non original image areas after rotation, such as color=100.
void Rotate(char* input,char* output,int angle)	Image rotation, where input is the input file name and output is the output file name. Supports BMP images. Angle is the angle of rotation.
void imgRotate90Gray(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	The grayscale image is rotated by 90.
void imgRotate90Color(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	Rotate the color image by 90 degrees.
void imgRotate270Gray(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	The grayscale image is rotated 270 degrees.

void imgRotate270Color(unsigned char *input,unsigned char *output,int sw,int sh,int *dw,int *dh)	Color image rotation 270.
void imgRotate180Gray(unsigned char *Img,int w,int h)	The grayscale image is rotated 180 degrees and the results are saved in the original input array.
void imgRotate180Color(unsigned char *Img,int w,int h)	The color image is rotated 180 degrees and the results are saved in the original input array.
void imgRBExchange(unsigned char *Img,int w,int h)	The color images R and B are interchangeable, and the results are saved in the original input array.
void NoiseUniform(char* input,char* output,double a,double b)	Uniformly distributed noise. Reference: a=0, b=0.2.
void NoiseGauss(char* input,char* output,float mean,float delta)	Gaussian noise. Reference: mean=0, delta=31.
void NoiseRayleigh(char* input,char* output,float a,float b)	Rayleigh noise. Reference: a=0, b=200.
void NoiseExp(char* input,char* output,float a)	Exponential noise. Reference: a=0.1.
void NoiseImpulse(char* input,char* output,float a,float b)	Spicy salt noise. Reference: a=0.2, b=0.2.
void grayToColor(FILE* input,FILE* output)	Grey to pseudo color, where input is the input file and output is the output file. Supports 8-bit and 24-bit BMP images.
void ImageThinning(char* input,char* output,char** str,int n,int m1,int a,int b)	Image refinement, where input is the input file name and output is the output file name. Supports 4-bit BMP images. n is the size of the first dimension of str, and m1 is the size of the second dimension, shaped like str [n] [m1]; a and b are related adjustment parameters, which can be a=3 and b=5. Reference template: 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)	Returns the minimum value of image pixels, where filename is the input image file name. Supports 8-bit and 24-bit BMP images.
int MaximumValueOfImagePixels(char* filename)	Returns the maximum value of image pixels, where filename is the input image file name. Supports 8-bit and 24-bit BMP images.
float AverageValueOfImagePixels(char* filename)	Returns the average value of image pixels, where filename is the input image file name. Supports 8-bit and 24-bit BMP images.
double StandardDeviationOfImagePixels(char* filename)	Returns the standard deviation of image pixels, where filename is the input image file name. Supports 8-bit and 24-bit BMP images.
double EntropyOfImage(char* filename)	Returns the entropy of the image, supporting 8-bit and 24-bit BMP images.
float* CountTheFrequencyOfPixels(char* filename)	filename is the name of the input image file. Store the frequency of each pixel, with pixel values ranging from 0 to 255. The element number in the return value array is the pixel value, and the value of this number under the array is the frequency of this pixel. Supports 8-bit and 24-bit BMP images.
void Rotate(char* input,char* output,int angle,int interpolation)	Image rotation. Reference: angle=80, interpolation=0, or interpolation=1.
void HSV(char* input,char* output,int h,int s,int v)	Image tone saturation and brightness adjustment, reference: h=120, s=60, v=20.
void ColorTransfer1(char* input1,char* input2,char* output)	Color transfer, supporting BMP images.

void OilpaintFilter(char* input,char* output,int radius,int smooth)	Oil filter. Reference: radius=10, smooth=100.
void HaloFilter(char* input,char* output,int ratio)	Halo angle filter. Reference: ratio=100.
void GrayHistogram(char* input,char* output,int hWidth,int hHeight)	Grayscale histogram. Reference: hWidth=256, hHeight=100.
void RedHistogram(char* input,char* output,int hWidth,int hHeight)	Red channel histogram. Reference: hWidth=256, hHeight=100.
void GreenHistogram(char* input,char* output,int hWidth,int hHeight)	Green channel histogram. Reference: hWidth=256, hHeight=100.
void BlueHistogram(char* input,char* output,int hWidth,int hHeight)	Blue channel histogram. Reference: hWidth=256, hHeight=100.
void HistogramEqualization2(char* input,char* output,int imgBit)	Histogram equalization: input is the name of the input file and output is the name of the output file. Supports 8-bit and 24-bit BMP images. imgBit is the number of digits in the input image.
void HistogramEqualization3(char* input,char* output)	Histogram equalization: input is the name of the input file and output is the name of the output file. Supports 8-bit and 24-bit BMP images.
void HistogramEqualization4(char* input,char* output)	Histogram equalization: input is the name of the input file and output is the name of the output file. Supports 8-bit and 24-bit BMP images. Input is the name of the input file, and out is the name of the output file.
void HistogramEqualization(char* input,char* output,int hWidth,int hHeight)	Histogram equalization. Reference: hWidth=256, hHeight=100.
void GrayHistogramEqualization(char* input,char* output,int hWidth,int hHeight)	Grayscale histogram. Reference: hWidth=256, hHeight=100.
void RedHistogramEqualization(cha	Red channel histogram. Reference: hWidth=256, hHeight=100.

r* input,char* output,int hWidth,int hHeight)	
void GreenHistogramEqualization(char* input,char* output,int hWidth,int hHeight)	Green channel histogram. Reference: hWidth=256, hHeight=100.
void BlueHistogramEqualization(char* input,char* output,int hWidth,int hHeight)	Blue channel histogram. Reference: hWidth=256, hHeight=100.
void GrayScaleStretch(char* input,char* output,int hWidth,int hHeight)	Grayscale stretching. Reference: hWidth=256, hHeight=100.
void GrayHistagramStretch(char* input,char* output,int hWidth,int hHeight)	Stretch the grayscale histogram. Reference: hWidth=256, hHeight=100.
void RedHistagramStretch(char* input,char* output,int hWidth,int hHeight)	Red channel histogram. Reference: hWidth=256, hHeight=100.
void GreenHistagramStretch(char* input,char* output,int hWidth,int hHeight)	Green channel histogram. Reference: hWidth=256, hHeight=100.
void BlueHistagramStretch(char* input,char* output,int hWidth,int hHeight)	Blue channel histogram. Reference: hWidth=256, hHeight=100.
void MedianFiltering1(char* input,char* output)	Median filtering, where input is the input file name and output is the output file name. Supports 8-bit BMP images.
void MedianFiltering2(char* input,char* output)	Median filtering, where input is the input file name and output is the output file name. Supports 8-bit and 24-bit BMP images.
void ThresholdProcessing(char* input,char* output,int Threshold)	Threshold processing, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Threshold is a threshold related parameter, such as Threshold=0.001.
void OTSUProcessing(char*	Otsu method processing, where input is

input,char* output)	the input file name and output is the output file name. Supports 8-bit BMP images.
void OBJtoTGA(char* input,char* output,int width,int height)	OBJ to TGA。
void ToRIM(char* input,char* output)	General images are transferred to RIM images, supporting PNG, JPG, and TGA images.
void ToImage(char* input,char* output,int jpg_quality)	RIM images are converted to general images, supporting PNG, JPG, and TGA images. jpg_quality=25。
void 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)	Convert a 1-bit deep monochrome BMP image into a bitmap print output of a thermal printer. The supported bitmap print instructions for the thermal printer are theESC *instructions. typedef unsigned char ARRAY3[3]; Reference: output="output. pbin", skip_ cmd = {0x1B, 0x4A, 0x00}, PRINTER_ TYPE_ BMP is the printer bitmap printing instruction code identifier, PRINTER_ TYPE_ BMP=(0x2A1B), mode is the printer bitmap printing mode, mode=33, FILE_ TYPE_ AD is an image type, and 'AD' represents an advertising image, FILE_TYPE_AD=(0x4441), a=0x80, b=1.
void WhiteBalance(const char* input,const char* output)	White balance.
void Sobel(char* input,char* output,double magnScale,double threshold)	Sobel operator, magnScale=0.35, threshold=130. Supports PGM and PBM images.
void Canny(char* input,char* output,double magnScale,double lowThreshold,double highThreshold)	Canny operator, magnScale=0.35, lowThreshold=55, highThreshold=120. Supports PGM and PBM images.
void BlackWhite(char* input,char* output,int threshold,int background)	Black and white, threshold=100, background=0. Supports PGM and PBM images.
void	Regional connectivity, threshold=100,

ConnectedComponents(char* input,char* output,int threshold,int background,int threshold1)	background=0, threshold 1=100. Supports PGM and PBM images.
void CleanImage(char* input,char* output)	Clean the image. Supports PGM and PBM images.
void NoiseImage(char* input,char* output,float probability)	Noise image, probability=0.1. Supports PGM and PBM images.
void HoughTransformCircle1(char* input,char* output,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1,double gamma1)	Circle detection. Scale1=1, gamma1=1.0, magnScale=0.5, lowThreshold=85, highThreshold=150, scale=0, gamma=1.0, sigma and kernelSize are used for smoothing Gaussian 5x5 kernels, sigma=1.0, kernelSize=5. If scale==0, the values remain unchanged, but if they are below 0 or above 255, they are set to 0 or 255, respectively. If scale= 0, then scale the value so that the minimum value is zero and the maximum value is 255. Set the gamma value to allow exponential scaling, and enable gamma=1.0. Supports PGM and PBM images.
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)	Circle detection. Scale1=1, gamma1=1.0, magnScale=0.5, lowThreshold=85, highThreshold=150, scale=0, gamma=1.0, sigma and kernelSize are used for smoothing Gaussian 5x5 kernels. sigma=1.0, kernelSize=5, number=10 indicates that the visual inspection of the image has 10 circles, minDist=35. If scale==0, the values remain unchanged, but if they are below 0 or above 255, they are set to 0 or 255, respectively. If scale= 0, then scale the value so that the minimum value is zero and the maximum value is 255. Set the gamma value to allow

	exponential scaling, and enable gamma=1.0. Supports PGM and PBM images.
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)	Circle detection. Scale1=1, gamma1=1.0, magnScale=0.5, lowThreshold=85, highThreshold=150, scale=0, gamma=1.0, sigma and kernelSize are used for smoothing Gaussian 5x5 kernels. sigma=1.0, kernelSize=5, number=10 indicates that the visual inspection of the image has 10 circles, minDist=35. If scale==0, the values remain unchanged, but if they are below 0 or above 255, they are set to 0 or 255, respectively. If scale= 0, then scale the value so that the minimum value is zero and the maximum value is 255. Set the gamma value to allow exponential scaling, and enable gamma=1.0. Returns elliptical data centered on (vCenter, hCenter) and radius (vradius, hradius), with a total of number sets of data, each containing one elliptical data. The first element is vCenter, the second element is hCenter, the third element is vradius, and the fourth element is hradius. Supports PGM and PBM images.
void ShapeEdgeDetection1(char* input,char* output,unsigned char CANNY_THRESH4,int CANNY_blur4)	Shape edge detection , CANNY_THRESH4=35 , CANNY_blur4=7. Supports PNG images.
void ShapeEdgeDetection2(char* input,char* output,unsigned char CANNY_THRESH4,int CANNY_blur4)	Shape edge detection , CANNY_THRESH4=35 , CANNY_blur4=7. Supports PNG images.
void ShapeEdgeDetection3(char* input,char* output,unsigned	Shape edge detection, CANNY_THRESH=50, CANNY_BLUR=12. Supports PNG images.

char CANNY_THRESH,int CANNY_BLUR)	
void ShapeEdgeDetection4(char* input,char* output,unsigned char CANNY_THRESH,int CANNY_BLUR)	Shape edge detection, CANNY_THRESH=50, CANNY_BLUR=12. Supports PNG images.
void ShapeEdgeDetection5(char* input,char* output,unsigned char CANNY_THRESH2,int CANNY_BLUR2)	Shape edge detection, CANNY_THRESH2=10, CANNY_BLUR2=2. Supports PNG images.
void ShapeEdgeDetection6(char* input,char* output,unsigned char CANNY_THRESH2,int CANNY_BLUR2)	Shape edge detection, CANNY_THRESH2=10, CANNY_BLUR2=2. Supports PNG images.
void ShapeEdgeDetection7(char* input,char* output,unsigned char CANNY_THRESH3,int CANNY_blur3)	Shape edge detection, CANNY_THRESH3=45, CANNY_blur3=10.Supports PNG images.
void ShapeEdgeDetection8(char* input,char* output,unsigned char CANNY_THRESH3,int CANNY_blur3)	Shape edge detection, CANNY_THRESH3=45, CANNY_blur3=10.Supports PNG images.

Other Processing

void Encode(char* input,char* output)	Text file compression, where input is the input file name and output is the output file name.
void Decode(char* input,char* output)	Decompress the text file compression result, where input is the input file name and output is the output file name.
void FileCompress(char *input , char *output)	File compression, where input is the input file name and output is the output file name.
void FileDecompression(char *input , char *output)	Decompress the file compression result, where input is the input file name and output is the output file name.

Advanced operator

void BlobAnalysis(char* input,char* output,int c1,int c2)	Blob analysis, c1 and c2 are color related parameters, reference: c1=128, c2=127. Supports BMP images.
void BlobAnalysis1(char* input,char* output,int c1,int c2)	Blob analysis, c1 and c2 are color related parameters, reference: c1=128, c2=127. Supports BMP images.
void VerificationCodeGeneration(char* inputText,char* output,int num,int foint,int a,int b1,int b2,int b3,int b4,int b5,int b6,int b7,int b8,int b9,int b10,int b11,int b12,int b13,double sigma,unsigned int noise_type,int width,int height,int depth,int spectrum,bool shared)	Verification code generation. sigma=10,noise_type=2,a=10,b1=128， b2=127，b3=2，b4=8，b5=12，b6=0，b7=1，b8=-100，b9=-100，b10=1，b11=3，b12=6，int b13=40，foint=30,num is the number of characters,depth=1,spectrum=3,shared=0。
void CornerDetection(char* input,char* output,float threshold, float k, float sigma,int width, int height, int channels)	Corner detection, threshold=10000, k=0.06, sigma=1.0, width=640, height=480, channels=1. Supports PNM images.
vector<Keypoint> CornerDetection1(char* input,char* output,float threshold, float k, float sigma,int width, int height, int channels)	Corner detection, returns corner data. threshold=10000，k=0.06，sigma=1.0，width=640，height=480，channels=1。 Supports PNM images. The following structures need to be introduced: 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)	Corner detection, returns corner data. threshold=2000, k=1, sigma=1.2。 Supports PNM images.
	The following namespace needs to be introduced:
	namespace Corner {
	struct Keypoint {
	float x;
	float y;
	float score;
	};
	}

void Structure(char* input,char* output,float sigma)	Feature normalization statistics, reference: sigma=2. Supports multiple image formats.
void Cornerness(char* input,char* output,float sigma,int method)	Corner detection, reference: sigma=2, method=0. Supports multiple image formats.
void Corners(char* input,char* output,float sigma, float thresh, int window, int nms, int corner_method)	Corner detection, reference: sigma=2, threshold=0.4, window=5, nms=3, corner_method=0。 Supports multiple image formats.
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	Feature matching, reference：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. Supports multiple image formats.

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)	

vector<Descriptor> HarrisCorner(char* input,char* output,float sigma1, float thresh1, int window1, int nms1, int corner_method1)	Corner detection, returns the detection results. Reference: sigma1=2, thresh1=0.4, window1=5, nms1=3, corner_method1=0。Supports multiple image formats.
	The following structures need to be introduced:
	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)	Describe the matching item and return the description result. Reference: sigma1=2, thresh1=0.4, window1=5, nms1=3, corner_method1=0 , sigma2=2 , thresh2=0.4 , window2=5, nms2=3, corner_method2=0。Supports multiple image formats.
	The following structures need to be introduced:
	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 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	Draw corner points. Reference: 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. Supports multiple image formats.

thresh6, int window6, int nms6, float inlier_thresh6, int iters6, int cutoff6, float acoeff6)	
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 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)	Make panoramic images. Reference: 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. Supports multiple image formats.
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	Corner detection. Reference: 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,

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,float sigma7, int corner_method7, float thresh7, int window7, int nms7, float inlier_thresh7, int iters7, int cutoff7, float acoeff7)	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. Supports multiple image formats.
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,	Corner detection. Reference: 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. Supports multiple image formats.

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,float sigma7, int corner_method7, float thresh7, int window7, int nms7, float inlier_thresh7, int iters7, int cutoff7, float acoeff7)	
int* FindLine(char* input,char* output,int width,int height)	Line detection, returning theta and rho of lines, supporting RAW images.
int* FindCircle(char* input,char* output,int width,int height,float sigma, int tmin, int tmax)	Circle detection, returns the coordinates of the center of the circle and the radius of the circle, supports RAW images.sigma=1.4 , tmin=70, tmax=150.
void TemplateMatching(char* input,char* Template,char* output,unsigned int b,double ps,double a,double a1,double b1,double c)	Template matching, a=0.5, b=0, a1=0.5, b1=0.5, c=0.2, ps is the similarity, such as ps=0.5. Supports 24 bit BMP images.
SearchResult TemplateMatching(char* input,char* Template)	Template matching returns the upper left corner coordinates and similarity values of the target location. Supports PNG images. The following structures need to be introduced: 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)	Template matching returns the upper left corner coordinates and similarity values of the target location. Supports PNG images.
	The following header file needs to be introduced:
	#define STB_IMAGE_IMPLEMENTATION
	#include "stb_image.h"
	#include <cstdint>
	#include <complex>
	#include <vector>
	Reference routine:
	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 ImageCalibration1(char* input,char* output)	Image calibration.

std::vector<Line> ImageCalibration2(char* input, char* output)	Image calibration.
	The following structures need to be introduced:
	struct Line {
	double m, b;
	int dist_o; // distance to origin point when
	// line(x = dist_o) is perpendicular to x axis
	int x0, x1, y0, y1; // two end points
	int end_point_num;
	Line(double _m, double _b, int _dist_o = 0, int _x0 = 0, int _y0 = 0,
	int _x1 = 0, int _y1 = 0, int _end_point_num = 0)
	: m(_m), b(_b), dist_o(_dist_o), x0(_x0), x1(_x1), y0(_y0), y1(_y1),
	end_point_num(_end_point_num) {}
	};

std::vector	Circle detection, returns the detection

<CentersPoint> FindCircles(char* input,char* output,int size1,int size2,int size3)	result. size1=5, size2=5, size3=7。 Supports BMP images.
	The following structures need to be introduced:
	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 ImgDisplay(char* input,char* WindowsName)	Image display, where input is the name of the input image file and WindowsName is the name of the window to be created.
vector<int> HoughTransform1(char* input,char* output, unsigned char color[3],int numOfLines,int fillGap,int minLength)	Hoff line detection, returns the detection result. If the gap between collinear segments is less than fillGap, connect them. If the merged row is shorter than minLength, discard it. numOfLines is the possible number of segments, numOfLines=10 , fillGap=5 , minLength=60, color={ 0, 255, 0 }. Supports 8-bit BMP images.
vector<int> HoughTransform2(char* input, char* output, unsigned char color[3],int numOfLines,int fillGap,int minLength)	Hoff line detection, returns the detection result. If the gap between collinear segments is less than fillGap, connect them. If the merged row is shorter than minLength, discard it. numOfLines is the possible number of segments, numOfLines=10 , fillGap=5 , minLength=60, color={ 0, 255, 0 }. Supports 8-bit BMP images.
void SobelGradient(char* input,char* output)	Use the sobel operator to calculate the gradients in the x and y directions respectively.
void SobelNoneMaxSpress(char* input, char* output)	Use the sobel operator to calculate the gradients in the x and y directions, and then use gradient direction and non maximum suppression to remove redundant information.

void HytheresisThresholding(char* input, char* output, double high, double low)	Hythereis threshold processing, high=130, low=60.
void HoughTransfer1(char* input, char* output, double high, double low, double threshold)	Hough transform, high=130, low=60, threshold=0.4.
void HoughTransfer3(char* input, char* output, double high, double low, double threshold)	Hough transform, high=130, low=60, threshold=0.4.
vector<int> HoughTransfer2(char* input, double high, double low, double threshold)	Hough transform returns corner information, consisting of two elements as a group. high=130, low=60, threshold=0.4.
void FeatureDetection1(char* input,char* output)	Feature detection.

std::vector<sift::Keypoint> FeatureDetection2(char* input, char* output)	Feature detection.
	The following space needs to be introduced:
	namespace sift
	{
	struct Keypoint {
	//Discrete coordinate system
	int i;
	int j;
	int octave;
	int scale; // Index of Gaussian Images in Octave Band
	
	// Continuous coordinates (interpolation)
	float x;
	float y;
	float sigma;
	float extremum_val; // Interpolated DoG extremum

	std::array<uint8_t, 128> descriptor;
	};
	}

void FeatureMatching(char* input1,char* input2,char* output)	Feature matching. Supports PGM images.
std::vector<std::pair<int, int>> FeatureMatching1(char* input1,char* input2,char* output)	Feature matching returns the matching result.
std::vector<std::pair<int, int>> FeatureMatching2(char* input1, char* input2, char* output)	Feature matching returns the matching result.
void Canny(char* input,char* output,int lowThreshold,int highThreshold)	The Canny operator supports at least JPG images, where input is the input file name and output is the output file name. Reference: lowThreshold=50, highThreshold=150.
void Canny(string input,string output)	Canny operator, reference: output="output". Supports BMP files.
void Canny(string input,char* output,float sigma, float threshold)	Canny operator, reference: sigma=6.0, threshold=3.5. Supports BMP files.
void Hough(char* input, char* output, float sigma, float threshold,double thre_val,unsigned char* color)	Hough transform, reference: sigma=6.0, threshold=3.5, thre_val=0.5, color is used to set the color of the drawn calibration points and lines, such as: color [3]={0,0,255}. Supports BMP files.
void DES_Encrypt(char *PlainFile, char *Key,char *CipherFile)	DES encryption function, supporting multiple files. PlainFile is the file name of the original file, Key is the key character, and CipherFile is the encrypted file name.
void DES_Decrypt(char *CipherFile, char *Key,char *PlainFile)	DES decryption function, supporting multiple files. CipherFile is the file name of the encrypted file, Key is the key character, and PlainFile is the decrypted file name.
int*	Template matching, where input is the parent

TemplateMatch(char* input,char* Template,char* output,int channels,int ROTATION)	image, Template is the sample image, and output is the file name of the resulting image. channels is the number of pixel channels in the image. The first element in the return value array is the maximum matching score, the second element is the X-axis coordinate value of the target, and the third element is the Y-axis coordinate value of the target. At least support PNG images. Reference: channels=3, ROTATION=360, or ROTATION=1.

int* TemplateMatch(image input,image Template,char* output,int channels,int ROTATION)	Template matching, where input is the parent image, Template is the sample image, and output is the file name of the resulting image. channels is the number of pixel channels in the image. The first element in the return value array is the maximum matching score, the second element is the X-axis coordinate value of the target, and the third element is the Y-axis coordinate value of the target. At least support PNG images. Reference: channels=3, ROTATION=360, or ROTATION=1.
	The following header files and structures need to be introduced:
	#include "stb_image.h"
	#include "stb_image_write.h"
	typedef struct imageContainer {
	int x,y,n;
	unsigned char *data;
	} image;
	Reference:
	image input, Template;
	input.data = stbi_load(inputFile, &input.x, &input.y, &input.n, 3); //3 indicates that the image has 3 pixel channels
	Template.data = stbi_load(templateFile, &Template.x, &Template.y, &Template.n, 3);

int* TemplateMatch(char* input,char* Template,int channels,int ROTATION)	Template matching, where input is the parent image and Template is the sample image. channels is the number of pixel channels in the image. The first element in the return value array is the maximum matching score, the second element is the X-axis coordinate value

	of the target, and the third element is the Y-axis coordinate value of the target. At least support PNG images. Reference: channels=3, ROTATION=360, or ROTATION=1.

int* TemplateMatch(image input,image Template,int ROTATION)	Template matching, where input is the parent image and Template is the sample image. The first element in the return value array is the maximum match score, the second element is the X-axis coordinate value of the target, and the third element is the Y-axis coordinate value of the target. At least support PNG images. Reference: ROTATION=360 or ROTATION=1.
	The following header files and structures need to be introduced:
	#include "stb_image.h"
	#include "stb_image_write.h"
	typedef struct imageContainer {
	int x,y,n;
	unsigned char *data;
	} image;
	Reference:
	image input, Template;
	input.data = stbi_load(inputFile, &input.x, &input.y, &input.n, 3); //3 indicates that the image has 3 pixel channels
	Template.data = stbi_load(templateFile, &Template.x, &Template.y, &Template.n, 3);

void KMeans(string input,unsigned int Clusters,char* output)	K-Means clustering, where input is the input file name, Clusters is the number of clusters, and output is the output file name. Supports BMP files.
void PGMSobel(char* input,char* output,int Mx[3][3],int My[3][3],int max,int min)	Sobel operator, where input is the input file name and output is the output file name. Supports PGM files in P5 format. Reference template: 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	X-direction filtering, where input is the input file name and output is the output file name. Supports PGM files in P5 format.

Mx[3][3],int My[3][3],int max,int min)	Reference template: 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-direction filtering, where input is the input file name and output is the output file name. Supports PGM files in P5 format. Reference template: 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 operator, where input is the input file name and output is the output file name. min and max are parameters related to image normalization, such as min=1000000, max=0; mx and my are the X and Y direction templates for Sobel operators, respectively. Supports PGM in P2 and P5 formats.
	Reference template:
	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 direction gradient, where input is the input file name and output is the output file name. min and max are parameters related to image normalization, such as min=1000000, max=0; mx and my are the X and Y direction templates for Sobel operators, respectively. Supports PGM in P2 and P5 formats. Reference template: 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 direction gradient, where input is the input file name and output is the output file name. min and max are parameters related to image normalization, such as min=1000000, max=0; mx and my are the X and Y direction templates for Sobel operators, respectively. Supports PGM in P2 and P5 formats.
	Reference template:
	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 operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format. XOutput is the gradient image in the X direction of the output, YOutput is the gradient image in the Y direction of the output, SobelOutput is the Sobel operator calculation result of the entire output image, and min and max are the relevant parameters for image normalization, such as min=100, max=0. Reference template: 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 operator, where input is the input file name and output is the output file name.

	Supports PGM files.
void Laplatian(char* input,char* output)	Laplacian operator, where input is the input file name and output is the output file name. Supports PGM files.
void HorizSobel(char* input,char* output)	Horizontal Sobel operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format.
void VertSobel(char* input,char* output)	Vertical Sobel operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format.
void PGMSobel1(char* input,char* output,int threshold)	Sobel operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format. Threshold is the target threshold, such as threshold=80.
void RAWSobelEdge(char* input,char* output,int ROWS,int COLS,int M,float sobelX[3][3],float sobelY[3][3])	Sobel operator, where input is the input file name and output is the output file name. ROWS is the row of the image, COLS is the column of the image, and M is the filtering related parameter, such as M=1. Support RAW images. Reference template: 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 RAWPlaceHolder(char* input,char* output,int ROWS,int COLS,int M,float mask[3][3])	Edge detection, where input is the input file name and output is the output file name. ROWS is the row of the image, COLS is the column of the image, and M is the filtering related parameter, such as M=1. Support RAW images. Reference template: float mask[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};
void LaplacianEnhancement (char* input,char* output,int N,int LaplMask[3][3])	Laplace image enhancement, where input is the input file name and output is the output file name. For example, N=1. Supports 8-bit BMP images. Reference template: int LaplMask[3][3] = { 0,1,0, 1, -4, 1, 0, 1, 0

	};

void LaplaceSmooth(char* input,char* output,int N,int LaplMask[3][3])	Laplace smoothing, where input is the input file name and output is the output file name. For example, N=1. Supports 8-bit BMP images.
	Reference template:
	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 operator, where input is the input file name and output is the output file name. For example, N=1. Supports 8-bit BMP images.
	Reference template:
	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 smoothing, where input is the input file name and output is the output file name. For example, N=1. Supports 8-bit BMP images.
	Reference template:
	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 Roberts(unsigned char** input,unsigned char** output)	Roberts operator, where input is the input data and output is the output data.
void Roberts(BMPMat** input,BMPMat** output)	Roberts operator, where input is the input data and output is the output data.

void SobelOperator(char* input,char* output)	The Sobel operator takes a long time, with input being the input file name and output being the output file name. Supports 24 bit BMP images.
SobelImage** SobelOperator(char* input)	Returns the coordinates and corresponding pixel values of each processed pixel point. If it is an edge point, it corresponds to white, otherwise it corresponds to black. Supports 24 bit BMP images. The following structures need to be introduced: typedef struct { int x; int y; unsigned char red; unsigned char green; unsigned char blue; }SobelImage;
void STLSection(char* input,char* output,int sliceAmount, int resolution,int c)	STL slicing, where input is the input STL file, output is the prefix name of the output slicing file, sliceAmount is the slicing amount, such as sliceAmount=50, resolution is the resolution, such as resolution=260, and c is the relevant parameter for execution, such as c=5.
void SURF(char* input1,char* input2,char* output)	SURF operator, input1 and input2 are input file names, and output is output file name, supporting BMP images.
void EdgeDetection(char* input,char* output)	Edge detection, where input is the input file name and output is the output file name. Supports 4-bit BMP images.
void EdgeDetection1(char* input,char* output,short sharpen[3][3])	Edge detection, where input is the input file name and output is the output file name. Supports 8-bit BMP images. Reference template: short sharpen[3][3] = {{1, 1, 1}, {1, -8, 1}, {1, 1, 1}};
void EdgeDetection2(char* input,char* output,int a)	Edge detection, where input is the input file name and output is the output file name. a is used to set the relevant parameters of image pixels, such as a=3. Supports 24 bit BMP images.
void	Edge detection, where input is the input file

EdgeDetection3(char* input,char* output,int a)	name and output is the output file name. a is used to set the relevant parameters of image pixels, such as a=3. Supports 24 bit BMP images.
void EdgeDetection4(char* input,char* output,int a)	Edge detection, where input is the input file name and output is the output file name. a is used to set the relevant parameters of image pixels, such as a=3. Supports 24 bit BMP images.
void YFiltering(char* input,char* output,int sobel_x[3][3],int sobel_y[3][3])	Y-direction filtering, where input is the input file name and output is the output file name. Supports PGM images in P5 format. Reference template: 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-direction filtering, where input is the input file name and output is the output file name. Supports PGM images in P5 format. Reference template: 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 operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format. Reference template: 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	The Prewitt operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format. Reference template: int prewitt_x[3][3] = { { 5, 5, 5},

prewitt_y[3][3])	{ -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])	The Laplace operator, where input is the input file name and output is the output file name. Supports PGM images in P5 format. Laplacian is a Laplacian operator template. Reference template: int laplacian[3][3] = { { 1, 1, 1}, { 1, -8, 1}, { 1, 1, 1}};
void Sobel0peration1(char* input,char* output,int width,int height)	Sobel operator, supporting RAW images.
void Sobel0peration2(char* input,char* output,int width,int height)	Sobel operator, supporting RAW images.
void Roberts(char* input,char* output)	Roberts edge detection, supporting BMP images.
void Prewitt(char* input,char* output)	Prewitt edge detection, supporting BMP images.
void Sobel(char* input,char* output)	Sobel operator, supporting BMP images.
void Laplace(char* input,char* output)	Laplace edge detection, supporting BMP images.
void BoxBlurAdvanced(string input,string output,int radius)	Advanced box blurring, reference: radius=5. Support PNG files.
void HoughTransform(char* input,char* output,unsigned char threshold)	Hough transform, where input is the input RAW file and output is the output RAS file, threshold=100.
static void EdgeDetectionWithoutNonmaximum(const LPCTSTR input, const	Edge detection, reference: a=0.33, b=0.33, c=0.33. Supports 24 bit BMP images.

LPCTSTR output,double a,double b,double c)	
static void CannyEdgeDetection(const LPCTSTR input, const LPCTSTR output,double a,double b,double c,int orank,int oranb)	Edge detection, reference: orange=20, orange=80. Supports 24 bit BMP images.
static void HoughTransform(const LPCTSTR input, const LPCTSTR output,double a,double b,double c,int orank,int oranb)	Hough transform, reference: a=0.33, b=0.33, c=0.33, orange=20, orange=80. Supports 24 bit BMP images.
void BoxBlurBasic(string input,string output)	The basic box is blurry and supports PNG files.
void SobelSharpen(char* input,char* output,int Templatex[3][3],int Templatey[3][3],int coefficient1,int coefficient2)	Sobel operator, where input is the input file name and output is the output file name. Templatex is the Laplace sharpening template with 4 neighborhoods, Templatey is the Laplace sharpening template with 8 neighborhoods, coefficent1=9, coefficent2=16. Supports 24 bit BMP images.
void EncryptionDecryption (char* input,char* output,int Key,int a)	Image encryption and decryption, where Key is the key, encryption is performed when a=1, and decryption is performed when a=0. Supports 24 bit BMP images.
void Encryption(char* input,char* output,int Key)	Image encryption, where input is the input file name and output is the output file name. Key is the key. Supports 24 bit BMP images.
void Decryption(char* input,char* output,int Key)	Image decryption, where input is the input file name and output is the output file name. Key is the key. Supports 24 bit BMP images.
void Nesting(char* Biginput,char* Smallinput,char* output)	Image nesting, Biginput is the large image of the input, and Smallinput is the small image of the input. Supports 24 bit BMP images.
void Blend(char* input1,char*	The blending of image fusion, input1 and input2 are the two input images to be fused,

input2,char* output)	and output is the output file name. Supports 24 bit BMP images.
void Checker(char* input1,char* input2,char* output)	The chessboard of image fusion, input1 and input2 are the two input images to be fused, and output is the output file name. Supports 24 bit BMP images.
void Blend1(char* input1,char* input2,char* output)	The blending of image fusion, input1 and input2 are the two input images to be fused, and output is the output file name. Supports 24 bit BMP images.
void Checker1(char* input1,char* input2,char* output)	The chessboard of image fusion, input1 and input2 are the two input images to be fused, and output is the output file name. Supports 24 bit BMP images.
void QRCodeGeneration(char *filename, char* inputString)	QR code generation, where filename is the name of the generated QR code image file and inputString is the information contained in the QR code. Supports BMP images.
vector<float> HarrisCornerDetection(char* input,int width,int height,int channels,int step,float threshold,float k,float sigma)	Corner detection, supporting PPM files in P5 and P6 formats. Starting from the first element in the return value array, the return values are grouped into three elements, namely the X coordinate, Y coordinate, and fraction of the corner. If the return value array is named A, {A [0], A [1], A [2]} are the data of the first corner, {A [3], A [4], A [5]} are the data of the second corner, and so on. Input is the name of the input image file, width and height are the width and height of the input image, channels are the number of channels in the input image, step defaults to -1, threshold is the score threshold for corners in Harris detection, k is the k value in Harris scoring function, and sigma is the sigma value used for IxIy array smoothing. Reference: threshold=2000, k=1, sigma=1.2.
vector<float> HarrisCorner(char* input,char* output,int width,int height,int channels,float threshold,float k,float sigma)	Corner detection, supporting PPM files in P5 and P6 formats. Starting from the first element in the return value array, the return values are grouped into three elements, namely the X coordinate, Y coordinate, and fraction of the corner. If the return value array is named A, {A [0], A [1], A [2]} are the data of the first corner, {A [3], A [4], A [5]} are

	the data of the second corner, and so on. Input is the name of the input image file, width and height are the width and height of the input image, channels are the number of channels in the input image, threshold is the score threshold for corners in Harris detection, k is the k value in Harris scoring function, and sigma is the sigma value used for IxIy array smoothing. Reference: threshold=2000, k=1, sigma=1.2.
int* TemplateMatching(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue,double MatchScore)	Template matching, the first element in the return value is the vertical coordinate of the top left corner of the matching box, the second element is the vertical coordinate of the bottom left corner of the matching box, the third element is the horizontal coordinate of the top left corner of the matching box, and the fourth element is the horizontal coordinate of the top right corner of the matching box. Input1 is the search image, input2 is the template image, output is the matching result image, and MatchScore=0.9. Supports BMP images.
int* TemplateMatching(char* input1,char* input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore)	Template matching, the first element in the return value is the vertical coordinate of the top left corner of the matching box, the second element is the vertical coordinate of the bottom left corner of the matching box, the third element is the horizontal coordinate of the top left corner of the matching box, and the fourth element is the horizontal coordinate of the top right corner of the matching box. Input1 is the search image, input2 is the template image, output is the matching result image, and MatchScore=0.9. Supports BMP images.
struct imagine TemplateMatching(struct imagine ColorSource,struct imagine input1,struct imagine input2,unsigned char red,unsigned char	Template matching returns the image of the matching result, input1 is the search image, input2 is the template image, and MatchScore=0.9. Supports BMP images. The following structures need to be introduced: typedef struct imagine { unsigned char *R,*G,*B,*header;

green,unsigned char blue,double MatchScore)	int W, H, Wpad, size;
	};
	State:
	void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie);
	struct imagine salvareBitmap (char *destinatieFisier);
	void afisare (struct imagine img, char *destinatieSalvare);
	Apply grayscale to all images:
	struct imagine output;
	grayscale_image(inputImage1, input1_grayscale);
	grayscale_image(inputImage2, input2_grayscale);
	ColorSource=salvareBitmap(inputImage1); //inputImage1 is a color original image
	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)	Template matching, the first element in the return value is the vertical coordinate of the top left corner of the matching box, the second element is the vertical coordinate of the bottom left corner of the matching box, the third element is the horizontal coordinate of the top left corner of the matching box, and the fourth element is the horizontal coordinate of the top right corner of the matching box. Input1 is the search image, input2 is the template image, output is the matching result image, and MatchScore=0.9. Supports BMP images.
	The following structures need to be introduced:
	typedef struct imagine {
	unsigned char *R,*G,*B,*header;
	int W, H, Wpad, size;
	};
	State:

	void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie);
	struct imagine salvareBitmap (char *destinatieFisier);
	Apply grayscale to all images:
	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 MaximumMatchingQuantity,double MatchScore,float suprapunereMaxima,unsigned char red,unsigned char green,unsigned char blue)	Template matching, superpunereMaxima represents the maximum overlap rate, reference: MaximumMatchingQuantity=10, MatchScore=0.8, superpunereMaxima=0.2. Supports BMP images.
int* TemplateMatching(char* input1,char* input2,char* output)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Supports BMP images.
int* TemplateMatching1(char* input1,char* input2)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Supports BMP images.
int* TemplateMatching2(char* input1,char* input2)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Supports BMP images.
my_image_comp* TemplateMatching(my_image_comp input,my_image_comp Template,int H,int length,float* hpf)	Template matching returns the matching result. Supports BMP images. The following structures need to be introduced: 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;
};
State:
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);
Reference:
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)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Supports BMP images.
	The following structures need to be introduced:
	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;
	};
	State:
	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);
	Reference：
	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)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Supports BMP images.
	The following structures need to be introduced：
	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;
	};
	State:
	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);
	Reference:
	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)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Min is a parameter related to the matching score and supports BMP images. Reference: min=65026.

int* TemplateMatching(bmp_in input1,bmp_in input2,float min)	Template matching, the return value is the coordinates (x, y) of the upper left corner of the matching box. Min is a parameter related to the matching score and supports BMP images. Reference: min=65026.
	The following structures need to be introduced:
	struct bmp_in {
	int num_components, rows, cols;
	int num_unread_rows;
	int line_bytes;
	int alignment_bytes;
	FILE *in;
	};
	State:
	extern int bmp_in__open(bmp_in *state, const

	char *fname);
	Reference:
	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)	Template matching, supporting JPG images. The first and second elements of the return value are the horizontal and vertical coordinates of the top left corner vertex of the matching box, the third element is the rotation angle of the target relative to the template, and the fourth element is the scaling ratio.
	Reference: size=1, best_loss=100000000,a=0.5, b=2.1, c=0.5, d=45, e1=20, e2=20.
	State:
	#define byte unsigned char
	byte ***LoadRgb(const char *fname, int *rows, int *cols, int *chan);
	Reference:
	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)	Template matching, supporting JPG images. The first and second elements of the return value are the horizontal and vertical coordinates of the top left corner vertex of the matching box, the third element is the rotation angle of the target relative to the template, and the fourth element is the scaling ratio.
	Reference: size=1, best_loss=100000000,a=0.5, b=2.1, c=0.5, d=45, e1=20, e2=20.
	State:
	#define byte unsigned char
	byte ***LoadRgb(const char *fname, int *rows, int *cols, int *chan);
	Reference:
	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)	Template matching, supporting JPG images, and returning matching results. Reference: size=1, best_loss=1000000000, a=0.5, b=2.1, c=0.5, d=45, e1=20, e2=20。
	State:
	#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);
	Reference:
	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)	Template matching returns the number of matched targets. Supports BMP images.
	The following header file needs to be introduced:
	#include "bmpread1.h"
	Reference:
	bmpread_t input, Template;
	bmpread(inputfile, BMPREAD_BYTE_ALIGN | BMPREAD_ANY_SIZE, &input);
	bmpread(Templatefile, BMPREAD_BYTE_ALIGN | BMPREAD_ANY_SIZE, &Template);

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)	Template matching, return value array: X and Y coordinates of the top left corner vertex of the matching box, width and height of the template, and degree of difference. Reference: output is the name of the matching result image, output_txt is a text file that stores matching related data, threshold=0.5, isWriteImageResult=1, color is the color of the matching box when the image is a grayscale image, and red, green, and blue are the red green blue channel values of the matching box color when the image is a color image. Support PPM files. The following structure needs to be introduced:

	typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; State： Image2* readPXM(const char* name); Reference： 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)	Template matching, return value array: X and Y coordinates of the top left corner vertex of the matching box, width and height of the template, and degree of difference. Reference: output_txt is a text file that stores matching related data, threshold=0.5, isWriteImageResult=1, color is the color of the matching box when the image is a grayscale image, and red, green, and blue are the red green blue channel values of the matching box color when the image is a color image. Support PPM files. The following structure needs to be introduced： typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; State： Image2* readPXM(const char* name); Reference： Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
double* TemplateMatching2(Image2* input, Image2* Template,char* output,char* output_txt,double	Template matching, return value array: X and Y coordinates of the top left corner vertex of the matching box, width and height of the template, and degree of difference. Reference: output is the name of the matching result image, output_txt is a text

threshold,int isWriteImageResult,unsigned char color,unsigned char red,unsigned char green,unsigned char blue)	file that stores matching related data, threshold=0.5, isWriteImageResult=1, color is the color of the matching box when the image is a grayscale image, and red, green, and blue are the red green blue channel values of the matching box color when the image is a color image. Support PPM files. The following structure needs to be introduced: typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; State: Image2* readPXM(const char* name); Reference: 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)	Template matching, return value array: X and Y coordinates of the top left corner vertex of the matching box, width and height of the template, and degree of difference. Reference: output_txt is a text file that stores matching related data, threshold=0.5, isWriteImageResult=1, color is the color of the matching box when the image is a grayscale image, and red, green, and blue are the red green blue channel values of the matching box color when the image is a color image. Support PPM files. The following structure needs to be introduced: typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; State: Image2* readPXM(const char* name);

	Reference: Image2* input = readPXM(inputFileName); Image2* Template = readPXM(templatename);
Image2* TemplateMatching3(Image2* input, Image2* Template, char* output_txt, double threshold, int isWriteImageResult, unsigned char color, unsigned char red, unsigned char green, unsigned char blue)	Template matching returns image data with matching results. Reference: output_txt is a text file that stores matching related data, threshold=0.5, isWriteImageResult=1, color is the color of the matching box when the image is a grayscale image, and red, green, and blue are the red green blue channel values of the matching box color when the image is a color image. The following structure needs to be introduced: typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2; State: Image2* readPXM(const char* name); Reference: 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)	Image matching, returns the image data of the matching result. Reference: output_txt is a text file that stores matching related data, threshold=0.5, isWriteImageResult=1, color is the color of the matching box when the image is a grayscale image, and red, green, and blue are the red green blue channel values of the matching box color when the image is a color image. The following structure needs to be introduced: typedef struct Image2 { int width; int height; int channel; unsigned char* data; }Image2;

	State: Image2* readPXM(const char* name); Reference: 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)	Template matching returns the center point coordinates, rotation angle, and scaling ratio of the matching box. Reference: c=0.5, threshold=0.9. Support PPM files.
	State:
	#ifndef __P
	#if defined(__STDC__) || defined(__cplusplus)
	#define __P(protos) protos
	#else
	#define __P(protos) ()
	#endif
	#endif
	RGB_PACKED_IMAGE *readRGBPackedImage __P((char*));
	The following structures need to be introduced:
	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;
	Reference:
	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	Template matching returns the center point coordinates, rotation angle, and scaling ratio of the matching box. Reference: c=0.5, threshold=0.9. Support PPM files. State: #ifndef __P #if defined(__STDC__) || defined(__cplusplus)

char green,unsigned char blue,double c,double threshold)	#define __P(protos) protos
	#else
	#define __P(protos) ()
	#endif
	#endif
	RGB_PACKED_IMAGE *readRGBPackedImage __P((char*));
	The following structures need to be introduced:
	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;
	Reference:
	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)	Template matching returns the center point coordinates, rotation angle, and scaling ratio of the matching box. Reference: c=0.5, threshold=0.9. Support PPM files. State: #ifndef __P #if defined(__STDC__) || defined(__cplusplus) #define __P(protos) protos #else #define __P(protos) () #endif #endif RGB_PACKED_IMAGE *readRGBPackedImage __P((char*)); The following structures need to be introduced: 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; Reference: 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)	Template matching returns image data with matching results. Reference: c=0.5, threshold=0.9. Support PPM files.
	State:
	#ifndef __P
	#if defined(__STDC__) || defined(__cplusplus)
	#define __P(protos) protos
	#else
	#define __P(protos) ()
	#endif
	#endif
	RGB_PACKED_IMAGE *readRGBPackedImage __P((char*));
	The following structures need to be introduced:
	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;
	Reference:
	RGB_PACKED_IMAGE* Template = readRGBPackedImage(templatename);
	RGB_PACKED_IMAGE* input = readRGBPackedImage(inputFileName);

int* TemplateMatching(char* input,char* Template)	Template matching, supports 24 bit BMP images, returns the upper left corner coordinates (x, y) of the matching box.

int* TemplateMatching(bmp_img input,bmp_img Template)	Template matching, supports 24 bit BMP images, returns the upper left corner coordinates (x, y) of the matching box.
	The following structures and declarations need to be introduced:
	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;
	State:
	enum bmp_error bmp_img_read(bmp_img *, const char *);
	Reference routine:
	bmp_img input, Template;
	bmp_img_read(&input, inputfile);
	bmp_img_read(&Template, outputfile);

int FeatureDetection(char* input,char* output,float a, int b,float red, float green, float blue)	Feature detection returns the number of found feature points. A is the brightness value, such as a=255, b is related to image clarity, and the default value is b=100; Red, green, and blue are the channel values of the red, green, and blue colors of the feature point indicator, such as red=0, green=0, and blue=1. Support PPM files.
unsigned int FeatureDetection(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue)	Feature detection returns the number of feature points. Support PPM files.

vector<int> FeatureMatching(char * input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue)	Feature matching, the data format in the return value is: the first value is the sequence number of the feature pair, starting from 0, the second and third values are the horizontal and vertical coordinates of one of the feature points in image 1, the fourth and fifth values are the horizontal and vertical coordinates of one of the feature points in image 2, and correspond to the feature points in image 1 where the second and third values are located. These five values form a group; Similarly, the sixth value is the sequence number of the next feature pair, which is 1, and so on for the subsequent values. Format such as: feature pair:%d, image1(%d,%d)->image2:(%d,%d). Support PPM files.
int FeatureMatching(char * input1,char* input2,char* output, float a, int b, float red, float green, float blue)	Feature matching returns the number of feature matching points found. A is the brightness value, such as a=255, b is related to image clarity, and the default value is b=100; Red, green, and blue are the channel values of red, green, and blue for the matching line color, such as red=0, green=0, and blue=1. Support PPM files.
unsigned int FeatureMatching1(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	Detect feature points from two input images, and then use brute force methods to match the features of the two images. input is the input image, output is the generated feature point image, bExtractDescriptor=true. Returns the number of matching feature points. Supports PGM files.
vector<int> FeatureMatching2(char* input1,char* input2,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	The data format in the return value is: the first value is the sequence number of the feature pair, starting from 0. The second and third values are the abscissa and ordinate of one of the feature points in image 1, respectively. The fourth and fifth values are the abscissa and ordinate of one of the feature points in image 2, corresponding to the feature point in image 1 where the second and third values are located. These five

	values form a group; Similarly, the sixth value is the sequence number of the next feature pair, which is 1, and so on for the subsequent values. Format such as: feature pair:%d, image1(%d,%d)->image2:(%d,%d). input is the input image, output is the generated feature point image, bExtractDescriptor=true. Supports PGM files.
unsigned int FeatureExtraction1(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	Returns the number of feature points. input is the input image, output is the generated feature point image, bExtractDescriptor=true. Supports PGM files.

std::list<ezsift::SiftKeypoint> FeatureExtraction2(char* input,char* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor)	Returns the list of feature points. input is the input image, output is the generated feature point image, bExtractDescriptor=true. Supports PGM files.
	The following namespace needs to be introduced:
	namespace ezsift {
	#define DEGREE (128) //SIFT Key Points: 128 Dimensions
	struct SiftKeypoint {
	int octave; //octave quantity
	int layer; // layer quantity
	float rlayer; // Actual quantity of layer
	float r; //Normalized row coordinates
	float c; //Normalized col coordinates
	float scale; //Normalized scale
	float ri; //Row coordinate (layer)
	float ci; //Column coordinates (layer)
	float layer_scale; // scale(layer)
	float ori; //degrees
	float mag; // Modulus
	float descriptors[DEGREE]; //descriptor
	};
	}

vector<int> DefectLocation(PGMData* Template,PGMData* Sample,int floor,int	Find the location of defects and return the location of material defects, in groups of every 4 elements. Template is the template image, Sample is the sample image, g is the

size,int a,int b,int c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report)	X-axis length of the effective limit of the defect, h is the Y-axis length of the effective limit of the defect, reference: 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.
	Introduce the following structure:
	typedef struct _PGMData {
	int row;
	int col;
	int max_gray;
	int **matrix;
	}PGMData;
	If the template file name is Template and the sample file name is Sample, use the following code to obtain the appropriate input data:
	First declare the readPGM function:
	PGMData* readPGM(const char *file_name, PGMData *data);
	Then execute the following code:
	PGMData* model = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Template,model);
	PGMData* data = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Sample,data);
	Afterwards, pass the model and data into the corresponding functions.
	Supports PGM files in P5 format.

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)	Defect size, returns the defect size in groups of 4. Template is the template image, Sample is the sample image, g is the X-axis length of the effective limit of the defect, h is the Y-axis length of the effective limit of the defect, reference: 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. Introduce the following structure: typedef struct _PGMData { int row; int col; int max_gray; int **matrix; }PGMData;

	If the template file name is Template and the sample file name is Sample, use the following code to obtain the appropriate input data:
	First declare the readPGM function:
	PGMData* readPGM(const char *file_name, PGMData *data);
	Then execute the following code:
	PGMData* model = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Template,model);
	PGMData* data = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Sample,data);
	Afterwards, pass the model and data into the corresponding functions.
	Supports PGM files in P5 format.

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)	The number of good or bad samples, in the returned results, the first element is the number of qualified circles, and the second element is the number of defective circles. Template is the template image, Sample is the sample image, g is the X-axis length of the effective limit of the defect, h is the Y-axis length of the effective limit of the defect, reference: 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.
	Introduce the following structure:
	typedef struct _PGMData {
	int row;
	int col;
	int max_gray;
	int **matrix;
	}PGMData;
	If the template file name is Template and the sample file name is Sample, use the following code to obtain the appropriate input data:
	First declare the readPGM function:
	PGMData* readPGM(const char *file_name, PGMData *data);
	Then execute the following code:
	PGMData* model = (PGMData*)malloc(sizeof(PGMData));
	readPGM(Template,model);

	PGMData* data = (PGMData*)malloc(sizeof(PGMData)); readPGM(Sample,data); Afterwards, pass the model and data into the corresponding functions. Supports PGM files in P5 format.

struct hough_param_circle* CircleDetection(char * input,int width,int height)	Circle detection returns relevant information such as the position and size of the found circle. Support RAW files.
	The following structures need to be introduced:
	struct hough_param_circle {
	int a;
	int b;
	int radius;
	int resolution;
	int thresh;
	struct point *points;
	int points_size;
	};

unsigned int* CircleDetection(char * input)	Circle detection returns the coordinates and radius of the center of a circle. The first element in the return value array is the X coordinate of the center, the second element is the Y coordinate of the center, and the third element is the radius of the circle. Supports BMP images.
int Equal(char* input1,char* input2,double c)	If the gradient similarity deviation value of the compared image is equal to c, it passes. input1 and input2 are the two images to compare. c is the reference threshold. Supports 24 bit BMP images.
int GreaterThan(char* input1,char* input2,double c)	If the gradient similarity deviation value of the compared image is greater than c, it is passed. input1 and input2 are the two images to compare. c is the reference threshold. Supports 24 bit BMP images.
int LessThan(char* input1,char* input2,double c)	If the gradient similarity deviation value of the compared image is less than c, it is passed. input1 and input2 are the two images to compare. c is the reference threshold. Supports 24 bit BMP images.
double GMSD(char* input1, char* input2)	Find the gradient similarity deviation value of two images and return the result. input1

	and input2 are the two images to compare. Supports 24 bit BMP images.
vector<vector<double>> Correction(string input,char* output, int A_height, int A_width,double thre_val, double sigma, double threshold, int angle_num, int range_thre, double fun_thre, int point_thre)	Image calibration, returning the coordinates of each vertex angle. Reference: A_height = 297, A_width = 210, thre_val=0.5, sigma=6.0, threshold=3.5, angle_num=180, range_thre=5, fun_thre=0.8, point_thre=3。 Supports BMP files.
vector<double> CalibrationAndCorrection(char* input,char* output1,char* output2, double sigma, double gra_threshold, double vote_threshold, double peak_dis,unsigned char* lines_color, unsigned char* intersections_color)	Image calibration returns the X-axis and Y-axis coordinates of the intersection points of each straight line. If the return value is received as vector<double> p, the coordinates of intersection 1 are (p[0],p[1]), intersection 2 is (p[2],p[3]), and so on. Reference: sigma=5.5, gra_threshold=30, vote_threshold=1000, peak_dis=200, lines_color[3]={0,255,0}, intersections_color[3]={0,0,255}. At least support BMP images.
float* ImageMatching(char* TargetImage,char* Template0,char* Template1,char* Template2,char* Template3,char* Template4,char* Template5,char* Template6,char* Template7,char* Template8,char* Template9)	Image matching, the first 10 elements in the return value are the difference scores between the target and the template in order, and the last element is the serial number of the matched template. Supports BMP images.
float* ImageMatching(char* TargetImage, char*	Image matching, the first two elements in the return value are the difference scores between the target and the template in order, and the

Template0, char* Template1)	last element is the serial number of the matched template. Supports BMP images.

float* ImageMatching(Image3 TargetImage,float **templates,int num_templates)	Image matching, if the number of templates is n, the first n elements in the return value are the difference scores between the target and the template in order, and the last element is the serial number of the matched template. Supports BMP images.
	The following structures need to be introduced:
	typedef struct
	{
	float *data;
	int width;
	int height;
	}Image3;
	
	State:
	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)	Image features. Sample kernel file content: 3 1 0 -1 0 -1 5 -1 0 -1 0 Among them, 3 represents the size of 3*3, and 1 represents the size of the kernel.
void FileWrite(char* BMP,char* TXT)	Write the image steganography file and write the text file to the image. Supports 32-bit BMP images. BMP is the name of the image file to be written, and TXT is the text file name of the image to be written.
void FileWriteOut(char* BMP,char* TXT)	Write the image steganography file and extract the text file from the image. Supports 32-bit BMP images. BMP is the name of the image file to be written out, and TXT is the name of the text file where the information is saved after the image is written out.
void LBP(char* input,char* output)	LBP image feature extraction. Supports PNG images.
void LBP(char* input,char* output, int choice, float radius,int pointNumbers)	LBP image feature extraction. Radius is the sampling radius (no less than 1 floating point number), pointNumbers is the number of sampling points (no less than 8 integers), and different values of choice represent: 1. Normal circular LBP 2. Rotation invariant circular LBP 3. Equivalent mode circular LBP. Supports 24 bit BMP images, and only in equivalent mode can the pointNumbers of sampling points be greater than 8 but not greater than 255.
void Watershed2(char* input,char* inputMarqueurs,char* output,int r,unsigned char R,unsigned char G,unsigned char B)	The watershed algorithm for image segmentation. InputMarqueurs is the labeled image of the input image. R=230, G=0, B=0, r=1。 Supports PNG images.
void EcrireImage1(char* input,char*	Image segmentation. rayon=5. Supports PNG images.

output,uint32_t rayon)	
void EcrireImage2(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	Image segmentation. rayon=5. Supports PNG images.
void EcrireLPECouleur1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	Image segmentation. rayon=5. Supports PNG images.
void Watershed1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	The watershed algorithm for image segmentation. inputMarqueurs is the labeled image of the input image. rayon=5. Supports PNG images.
void EcrireImage3(char* input,char* inputMarqueurs,char* output,uint16_t rayon)	Image segmentation. rayon=1. Supports PNG images.
void EcrireImageCouleursAleatoires(char* input,char* inputMarqueurs,char* output,uint8_t r,uint8_t g,uint8_t b,uint16_t rayon)	Image segmentation. rayon=1. Supports PNG images.
void Watershed(char* input,char* inputMarqueurs,char* output,uint8_t r,uint8_t g,uint8_t b,uint8_t a,uint16_t rayon)	The watershed algorithm for image segmentation. inputMarqueurs is the labeled image of the input image. a is usually 255, and rayon=1. Supports PNG images.
void FloodFill(char* input,char* output,int x,int y,unsigned char	Overflow filling method for image segmentation. x=0, y=0, novaCor=127。 Supports PGM files.

novaCor)	
void ConvertCoordinatesToGraphics(char* input,char* output, double IMAGE_SIZE, double PIXEL_PADDING, bool drawlines)	Read point coordinates and output images of points or line segments drawn between points. IMAGE_SIZE=800 , PIXEL_PADDING=25, drawlines=0, or drawlines=1. Input file format If we represent 'N' as a number of points, then assume the following point coordinate file format: N 1 x coordinate y coordinate 2 x coordinate y coordinate 3 x coordinate y coordinate 4 x coordinate y coordinate .... N x coordinate y coordinate
void HumanDetection1(char* input,char* output, double MINH, double MAXH, double MINS, double MAXS)	Human detection. Reference: MINH=0.0, MAXH=50.0, MINS=0.23, MAXS=0.68. Supports 24 bit BMP images.
void HumanDetection2(char* input, char* output, double MINH, double MAXH, double MINS, double MAXS)	Human detection. Reference: MINH=0.0, MAXH=50.0, MINS=0.23, MAXS=0.68. Supports 24 bit BMP images.
void HumanDetection3(char* input, char* output, double MINH, double MAXH, double MINS, double MAXS)	Human detection. Reference: MINH=0.0, MAXH=50.0, MINS=0.23, MAXS=0.68. Supports 24 bit BMP images.
int ImageFeatureNumber(char* input)	Calculate the number of feature points in the image. Supports 24 bit BMP images.
int ContentSimilarity(char* input1, char* input2,int KDTREE_BBF_MAX_NN_CHKS,double NN_SQ_DIST_RATIO_THR	Returns the content similarity between two images. KDTREE_BBF_MAX_NN_CHKS is the maximum number of key point NN candidates to be checked during BBF search, NN_SQ_DIST_RATIO_THR is the threshold of the squared distance ratio between NN and the second NN, reference :

)	KDTREE_BBF_MAX_NN_CHKS=100 , NN_SQ_DIST_RATIO_THR=0.49. Supports 24 bit BMP images.

Feature* ImageFeature(char* input)	Returns the feature data of the image. Supports 24 bit BMP images.
	The following structure needs to be introduced:
	typedef struct Point2D64f
	{
	double x;
	double y;
	}Point2D64f;
	
	typedef struct feature
	{
	double x; //x coord
	double y; //y coord
	double a; //Oxford-type affine region parameter
	double b; //Oxford-type affine region parameter
	double c; //Oxford-type affine region parameter
	double scl; //scale of a Lowe-style feature
	double ori; //orientation of a Lowe-style feature
	int d; //descriptor length
	double descr[FEATURE_MAX_D]; //descriptor
	int type; //feature type, OXFD or LOWE
	int category; //all-purpose feature category
	struct feature* fwd_match; //matching feature from forward image
	struct feature* bck_match; //matching feature from backmward image
	struct feature* mdl_match; //matching feature from model
	Point2D64f img_pt; //location in image
	Point2D64f mdl_pt; //location

	in model
	void* feature_data; //user-definable data
	}Feature;

double CharacterRecognition (char* TargetImage,char* TemplateFileGroup[])	Character matching, supports BMP images, and the return value is the sequence number of the template file matched to the target image. If the return value is 2, it indicates that the image matches the template with sequence number 2 (starting from zero). Reference : TemplateFileGroup[]={ "0.txt", "1.txt", "2.txt", "3.txt", "4.txt", "5.txt", "6.txt", "7.txt", "8.txt", "9.txt" };
double CharacterRecognition1(char* TargetImage,char* TemplateFileGroup[])	Character matching, supports BMP images, and the return value is the sequence number of the template file matched to the target image. If the return value is 2, it indicates that the image matches the template with sequence number 2 (starting from zero). Reference : TemplateFileGroup[]={ "0.txt", "1.txt", "2.txt", "3.txt", "4.txt", "5.txt", "6.txt", "7.txt", "8.txt", "9.txt" };