User Manual

image_t* ReadPNM(char* input)	Read PNM files, supporting PBM, PGM, and PPM images. The following structures need to be introduced: typedef struct color_t { uint8_t r; //Red uint8_t g; //Green uint8_t b; //Blue uint8_t a; //Alpha } color_t; typedef union pixcel_t { color_t c; //RGBA uint8_t g; //Grayscale uint8_t i; //Color index } pixcel_t; typedef struct image_t { uint32_t width; uint32_t height; uint16_t color_type; uint16_t palette_num; color_t *palette; // Pointer to the palette pixcel_t **map; //Image data } image_t;
void WritePNM(image_t* input,char* output,int type)	PNM image data is saved as an image file, supporting PBM, PGM, and PPM images. Type is the format of the PNM file, such as type=1, 2, 3, 4, 5, and 6. The following structures need to be introduced: typedef struct color_t { uint8_t r; //Red uint8_t g; //Green uint8_t b; //Blue uint8_t a; //Alpha } color_t; typedef union pixcel_t { color_t c; //RGBA uint8_t g; //Grayscale uint8_t i; //Color index } pixcel_t; typedef struct image_t {

	uint32_t width;
	uint32_t height;
	uint16_t color_type;
	uint16_t palette_num;
	color_t *palette; // Pointer to the palette
	pixcel_t **map; //Image data
	} image_t;

image_t* ReadBMP(char* input)	Read BMP images.
	The following structures need to be introduced:
	typedef struct color_t {
	uint8_t r; //Red
	uint8_t g; //Green
	uint8_t b; //Blue
	uint8_t a; //Alpha
	} color_t;
	typedef union pixcel_t {
	color_t c; //RGBA
	uint8_t g; //Grayscale
	uint8_t i; //Color index
	} pixcel_t;
	typedef struct image_t {
	uint32_t width;
	uint32_t height;
	uint16_t color_type;
	uint16_t palette_num;
	color_t *palette; // Pointer to the palette
	pixcel_t **map; //Image data
	} image_t;

void WriteBMP(image_t* input,char* output,int compress)	BMP image data is saved as an image file, and RLE compress is performed when compress=1.
	The following structures need to be introduced:
	typedef struct color_t {
	uint8_t r; //Red
	uint8_t g; //Green
	uint8_t b; //Blue
	uint8_t a; //Alpha
	} color_t;
	typedef union pixcel_t {
	color_t c; //RGBA

	uint8_t g; //Grayscale
	  uint8_t i; //Color index
	} pixcel_t;
	typedef struct image_t {
	  uint32_t width;
	  uint32_t height;
	  uint16_t color_type;
	  uint16_t palette_num;
	  color_t *palette; // Pointer to the palette
	  pixcel_t **map; //Image data
	} image_t;

void WriteBMP(image_t* input,char* output)	BMP image data is saved as an image file.
	The following structures need to be introduced:
	typedef struct color_t {
	  uint8_t r; //Red
	  uint8_t g; //Green
	  uint8_t b; //Blue
	  uint8_t a; //Alpha
	} color_t;
	typedef union pixcel_t {
	  color_t c; //RGBA
	  uint8_t g; //Grayscale
	  uint8_t i; //Color index
	} pixcel_t;
	typedef struct image_t {
	  uint32_t width;
	  uint32_t height;
	  uint16_t color_type;
	  uint16_t palette_num;
	  color_t *palette; // Pointer to the palette
	  pixcel_t **map; //Image data
	} image_t;

void WriteBMP1(image_t* input,char* output,int compress)	BMP image data is saved as an image file, and RLE compress is performed when compress=1.
	The following structures need to be introduced:
	typedef struct color_t {
	  uint8_t r; //Red
	  uint8_t g; //Green

	uint8_t b; //Blue
	uint8_t a; //Alpha
	} color_t;
	typedef union pixcel_t {
	color_t c; //RGBA
	uint8_t g; //Grayscale
	uint8_t i; //Color index
	} pixcel_t;
	typedef struct image_t {
	uint32_t width;
	uint32_t height;
	uint16_t color_type;
	uint16_t palette_num;
	color_t *palette; // Pointer to the palette
	pixcel_t **map; //Image data
	} image_t;

void ImageFusion(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 ImageFusion(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=″ImageFusion1.jpg″, input2=″ImageFusion2.jpg″, MaskImage=″Mask.png″, output=″output.jpg″。
	int dx[] = {0,0,-1,1};
	int dy[] = {-1,1,0,0};

void ImageFusion(char* input1,char* input2,char* inputUniqe1,char* inputUniqe2,char* output)	Image fusion, supporting PNG images. reference : input1=”ImageFusion1.png”, input2=”ImageFusion2.png”, inputUniqe1=”ImageFusion1_unique.txt”, inputUniqe2=”ImageFusion2_unique.txt”。
void Uniqe(char* input,char* inputUniqe,char* output,double R,double G,double B)	Image fusion, supporting PNG images. reference : input=”ImageFusion1.png”, inputUniqe=”ImageFusion1_unique.txt”。R=255, G=0, B=0。
void Screenshot1(HWND hWnd, LPCWSTR OutputImage)	Screenshot function. hWnd is the window handle to be screenshot, such as : GetDesktopWindow() ;

	OutputImage is the name of the screenshot.
void Screenshot2(HWND hWnd,LPCWSTR OutputImage)	Screenshot function. hWnd is the window handle to be screenshot, such as : GetDesktopWindow() ; OutputImage is the name of the screenshot.
void Screenshot3(HWND hWnd, LPCWSTR OutputImage)	Screenshot function. hWnd is the window handle to be screenshot, such as : GetDesktopWindow() ; OutputImage is the name of the screenshot.
uint8_t* AESencrypt(uint8_t* input,uint8_t* key,int size)	AES encryption function, where input is the original data, key is the key, and size is the size of the input. Return encrypted result data.
uint8_t* AESdecrypt(uint8_t* input,uint8_t* key,int size)	AES decryption function, where input is encrypted data, key is the key, and size is the size of the input. Return decryption result data.
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 Equal(char* input1,char* input2,double c)	If the similarity deviation value of the gradient amplitude of the compared image is equal to c, it is passed. Input1 and input2 are two images to be compared. c is the reference threshold. Supports 24 bit BMP images.
int GreaterThan(char* input1,char* input2,double c)	If the similarity deviation value of the gradient amplitude of the compared image is greater than c,

	it is passed. Input1 and input2 are two images to be compared. c is the reference threshold. Supports 24 bit BMP images.
int LessThan(char* input1,char* input2,double c)	If the gradient amplitude similarity deviation value of the compared image is less than c, it is passed. Input1 and input2 are two images to be compared. c is the reference threshold. Supports 24 bit BMP images.
double GMSD(char* input1, char* input2)	Find the gradient similarity deviation value between two images and return the result. Input1 and input2 are two images to be compared. Supports 24 bit BMP images.
void FileWrite(char* BMP,char* TXT)	Write the image steganography file and write the text file into the image. Supports 32-bit BMP images. BMP is the file name of the image 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 image file name to be written, and TXT is the text file name where the information is saved after writing the image.
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 annotated image of the input image. R=230, G=0, B=0, r=1. Supports 24 bit BMP images.
void EcrireImage1(char* input,char* output,uint32_t rayon)	Image segmentation. rayon=5. Supports 24 bit BMP images.
void EcrireImage2(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	Image segmentation. rayon=5. Supports 24 bit BMP images.
void EcrireLPECouleur1(char*	Image segmentation. rayon=5.

input,char* inputMarqueurs,char* output,uint32_t rayon)	Supports 24 bit BMP images.
void Watershed1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	The watershed algorithm for image segmentation. inputMarqueurs is the annotated image of the input image. rayon=5. Supports 24 bit BMP images.
void EcrireImage3(char* input,char* inputMarqueurs,char* output,uint16_t rayon)	Image segmentation. rayon=1. Supports 24 bit BMP 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 24 bit BMP 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 annotated image of the input image. a is generally 255, and rayon=1. Supports 24 bit BMP images.
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″ };
void CharacterSegmentation(char* input, string OutputFolder, int YHistogramValleyMaxPixelNumber, int XHistogramValleyMaxPixelNumber, double SubImgBlackPixelPercentage, int SingleNumberImgBoundary, int Infinite, double NumberImageBlackPixelPercentage)	Character segmentation. Supports BMP images. OutputFolder is the folder where the results are output, such as ″output″. The file name for the output results is composed of: X coordinate in the top left corner - Y coordinate in the top left corner - X coordinate in the bottom right corner - Y coordinate in the bottom right corner, YHistogramValleyMaxPixelNumber is the minimum number of black pixels in the valley of the Y-direction histogram, YHistogramValleyMaxPixelNumber=0, XHistogramValleyMaxPixelNumber is the minimum number of black pixels in the valley of the X-direction histogram, XHistogramValleyMaxPixelNumber=4， SubImgBlackPixelPercentage is the percentage of black pixels in a subgraph that is considered a number, SubImgBlackPixelPercentage=0.001， SingleNumberImgBoundary is the edge fill width of a single digital image, SingleNumberImgBoundary=5，Infinite is considered infinite , Infinite=249480 , NumberImageBlackPixelPercentage is the number of black pixels in a single digital image that exceeds all digital images, NumberImageBlackPixelPercentage=0.35。

void CharacterSegmentation(char* input,char* output, int BoundaryRemoveGap, int BinaryGap,int YHistogramValleyMaxPixelNumber, double SubImgBlackPixelPercentage, int Infinite, int XHistogramValleyMaxPixelNumber, double NumberImageBlackPixelPercentage, int SingleNumberImgBoundary)	Character segmentation. Supports BMP images. BinaryGap is the global threshold for image binarization ， BinaryGap=135，BoundaryRemoveGap is the distance where all edges are set to white，BoundaryRemoveGap=7，Infinite is considered infinite，Infinite=249480 ， SingleNumberImgBoundary is the edge fill width of a single digital image，SingleNumberImgBoundary=5，YHistogramValleyMaxPixelNumber is the minimum number of black pixels in the valley of the Y-direction histogram ， YHistogramValleyMaxPixelNumber=0，XHistogramValleyMaxPixelNumber is the minimum number of black pixels in the valley of the X-direction histogram ， XHistogramValleyMaxPixelNumber=4，SubImgBlackPixelPercentage is the percentage of black pixels in a subgraph that is considered a number ， SubImgBlackPixelPercentage=0.001，NumberImageBlackPixelPercentage is the number of black pixels in a single digital image that exceeds all digital images ， NumberImageBlackPixelPercentage=0.35。 Reference：output="output"。

void CodeEncoding(std::string input,char* output, int width,int height, int margin, int eccLevel, int stride_bytes, int comp,int a)	QR code encoding. input is the string to be encoded, and output is the file name of the generated QR code image.
	Margin: The margin around the barcode
	ECC: Error correction level，[0-8]
	a=1：AZTEC
	a=2：CODABAR
	a=3：CODE_39

	a=4：CODE_93
	a=5：CODE_128
	a=6：DATA_MATRIX
	a=7：EAN_8
	a=8：EAN_13
	a=9：ITF
	a=10：MAXICODE
	a=11：PDF_417
	a=12：QR_CODE
	a=13：RSS_14
	a=14：RSS_EXPANDED
	a=15：UPC_A
	a=16：UPC_E
	a=17：UPC_EAN_EXTENSION
	Reference：margin=10，eccLevel=-1，stride_bytes=0，comp=1。

std::string CodeDecoding(char* input,int req_comp,int a)	QR code decoding. input is the file name of the input QR code image, and returns the decoding result.
	a=1：Lum
	a=2：RGB
	a=3：BGR
	a=4：RGBX
	a=5：XRGB
	a=6：BGRX
	a=7：XBGR
	Reference：req_comp=4，a=4。