Manual do utilizador

image_t* ReadPNM(char* input)	Leia arquivos PNM, suportando imagens PBM, PGM e PPM.
	É necessário introduzir as seguintes estruturas:
	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; //Tons de Cinzento
	uint8_t i; //Índice de Cores
	} pixcel_t;
	typedef struct image_t {
	uint32_t width; //largo
	uint32_t height; //alta
	uint16_t color_type; //Categoria de cores
	uint16_t palette_num; //Número de paletas de cores
	color_t *palette; //Apontador para a paleta
	pixcel_t **map; //dados da imagem
	} image_t;

void WritePNM(image_t* input,char* output,int type)	Os dados de imagem PNM são salvos como um arquivo de imagem, suportando imagens PBM, PGM e PPM. Tipo é o formato do arquivo PNM, como tipo=1, 2, 3, 4, 5 e 6.
	É necessário introduzir as seguintes estruturas:
	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; //Tons de Cinzento

	uint8_t i; //Índice de Cores
	} pixcel_t;
	typedef struct image_t {
	uint32_t width; //largo
	uint32_t height; //alta
	uint16_t color_type; //Categoria de cores
	uint16_t palette_num; //Número de paletas de cores
	color_t *palette; //Apontador para a paleta
	pixcel_t **map; //dados da imagem
	} image_t;

image_t* ReadBMP(char* input)	Leia imagens BMP.
	É necessário introduzir as seguintes estruturas:
	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; //Tons de Cinzento
	uint8_t i; //Índice de Cores
	} pixcel_t;
	typedef struct image_t {
	uint32_t width; //largo
	uint32_t height; //alta
	uint16_t color_type; //Categoria de cores
	uint16_t palette_num; //Número de paletas de cores
	color_t *palette; //Apontador para a paleta
	pixcel_t **map; //dados da imagem
	} image_t;

void WriteBMP(image_t* input,char* output,int compress)	Os dados de imagem BMP são salvos como um arquivo de imagem, e a compressão RLE é executada quando compress=1.

	É necessário introduzir as seguintes estruturas:
	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; //Tons de Cinzento
	  uint8_t i; //Índice de Cores
	} pixcel_t;
	typedef struct image_t {
	  uint32_t width; //largo
	  uint32_t height; //alta
	  uint16_t color_type; //Categoria de cores
	  uint16_t palette_num; //Número de paletas de cores
	  color_t *palette; //Apontador para a paleta
	  pixcel_t **map; //dados da imagem
	} image_t;

void WriteBMP(image_t* input,char* output)	Os dados de imagem BMP são salvos como um arquivo de imagem.
	É necessário introduzir as seguintes estruturas:
	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; //Tons de Cinzento
	  uint8_t i; //Índice de Cores
	} pixcel_t;
	typedef struct image_t {
	  uint32_t width; //largo
	  uint32_t height; //alta
	  uint16_t color_type;

	//Categoria de cores
	uint16_t palette_num; //Número de paletas de cores
	color_t *palette; //Apontador para a paleta
	pixcel_t **map; //dados da imagem
	} image_t;

void WriteBMP1(image_t* input,char* output,int compress)	Os dados de imagem BMP são salvos como um arquivo de imagem, e a compressão RLE é executada quando compress=1.
	É necessário introduzir as seguintes estruturas:
	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; //Tons de Cinzento
	uint8_t i; //Índice de Cores
	} pixcel_t;
	typedef struct image_t {
	uint32_t width; //largo
	uint32_t height; //alta
	uint16_t color_type; //Categoria de cores
	uint16_t palette_num; //Número de paletas de cores
	color_t *palette; //Apontador para a paleta
	pixcel_t **map; //dados da imagem
	} image_t;

void ImageFusion(char* input1,char* input2,char* output,int block_height,int block_width,double threshold)	Fusão de imagens multi foco, suportando imagens BMP de 8 bits. block_height=8, block_width=8, threshold=1,75.
void ImageFusion(char* input1,char* input2,char* MaskImage,char* output,int	Fusão de imagens. referência: a=3, b1=4, DX1=-68, DY1=-99, EPS=1, input1=" Fusão de imagens 1.jpg",

dx[],int dy[],int a,double b1,int DX1,int DY1,double EPS)	input2=" Fusão de imagens 2.jpg", MaskImage=" Máscara.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)	Fusão de imagens, suportando imagens PNG. referência：input1=" Fusão de imagens 1.png", input2=" Fusão de imagens 2.png" , inputUniqe1=" Fusão de imagens 1_unique.txt" , inputUniqe2=" Fusão de imagens 2_unique.txt"。
void Uniqe(char* input,char* inputUniqe,char* output,double R,double G,double B)	Fusão de imagens, suportando imagens PNG. referência：input=" Fusão de imagens 1.png" , inputUniqe=" Fusão de imagens 1_unique.txt"。R=255, G=0, B=0。
void Screenshot1(HWND hWnd, LPCWSTR OutputImage)	Função de captura de ecrã. hWnd é a alça da janela para ser captura de tela, como：GetDesktopWindow(); OutputImage é o nome da captura de tela.
void Screenshot2(HWND hWnd,LPCWSTR OutputImage)	Função de captura de ecrã. hWnd é a alça da janela para ser captura de tela, como：GetDesktopWindow(); OutputImage é o nome da captura de tela.
void Screenshot3(HWND hWnd, LPCWSTR OutputImage)	Função de captura de ecrã. hWnd é a alça da janela para ser captura de tela, como：GetDesktopWindow(); OutputImage é o nome da captura de tela.
uint8_t* AESencrypt(uint8_t* input,uint8_t* key,int size)	Função de criptografia AES, onde input é os dados originais, key é a chave e size é o tamanho de input. Devolve os dados encriptados do resultado.
uint8_t* AESdecrypt(uint8_t* input,uint8_t* key,int size)	Função de descriptografia AES, onde input é os dados criptografados, key é a chave, e size é o tamanho de input. Devolve os dados do resultado da descriptografia.
void DES_Encrypt(char *PlainFile, char *Key,char	Função de encriptação DES, suportando vários ficheiros.

*CipherFile)	PlainFile é o nome do arquivo original, Key é o caractere chave e CipherFile é o nome do arquivo criptografado.
void DES_Decrypt(char *CipherFile, char *Key,char *PlainFile)	DES função de descriptografia, suportando vários arquivos. CipherFile é o nome do arquivo criptografado, Key é o caractere de chave e PlainFile é o nome do arquivo descriptografado.
int Equal(char* input1,char* input2,double c)	Se o valor do desvio de similaridade da amplitude do gradiente da imagem comparada for igual a c, ele é passado. Input1 e input2 são duas imagens a serem comparadas. c é o limiar de referência. Suporta imagens BMP de 24 bits.
int GreaterThan(char* input1,char* input2,double c)	Se o valor do desvio de similaridade da amplitude do gradiente da imagem comparada for maior que c, ele é passado. Input1 e input2 são duas imagens a serem comparadas. c é o limiar de referência. Suporta imagens BMP de 24 bits.
int LessThan(char* input1,char* input2,double c)	Se o valor do desvio de similaridade da amplitude do gradiente da imagem comparada for menor que c, ele é passado. Input1 e input2 são duas imagens a serem comparadas. c é o limiar de referência. Suporta imagens BMP de 24 bits.
double GMSD(char* input1, char* input2)	Encontre o valor de desvio de similaridade gradiente entre duas imagens e retorne o resultado. input1 e input2 são duas imagens a serem comparadas. Suporta imagens BMP de 24 bits.
void FileWrite(char* BMP,char* TXT)	Escreva o arquivo de esteganografia da imagem e escreva o arquivo de texto na imagem. Suporta imagens BMP de 32 bits. BMP é o nome do

	arquivo da imagem a ser escrita, e TXT é o nome do arquivo de texto da imagem a ser escrita.
void FileWriteOut(char* BMP,char* TXT)	Escreva o arquivo de esteganografia da imagem e extraia o arquivo de texto da imagem. Suporta imagens BMP de 32 bits. BMP é o nome do arquivo de imagem a ser escrito, e TXT é o nome do arquivo de texto onde as informações são salvas após a gravação da imagem.
void Watershed2(char* input,char* inputMarqueurs,char* output,int r,unsigned char R,unsigned char G,unsigned char B)	O algoritmo divisor de águas para segmentação de imagens. inputMarqueurs é a imagem anotada da imagem de entrada. R=230, G=0, B=0, r=1. Suporta imagens BMP de 24 bits.
void EcrireImage1(char* input,char* output,uint32_t rayon)	Segmentação de imagens. rayon=5. Suporta imagens BMP de 24 bits.
void EcrireImage2(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	Segmentação de imagens. rayon=5. Suporta imagens BMP de 24 bits.
void EcrireLPECouleur1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	Segmentação de imagens. rayon=5. Suporta imagens BMP de 24 bits.
void Watershed1(char* input,char* inputMarqueurs,char* output,uint32_t rayon)	O algoritmo divisor de águas para segmentação de imagens. inputMarqueurs é a imagem anotada da imagem de entrada. rayon=5. Suporta imagens BMP de 24 bits.
void EcrireImage3(char* input,char* inputMarqueurs,char* output,uint16_t rayon)	Segmentação de imagens. rayon=1. Suporta imagens BMP de 24 bits.
void EcrireImageCouleursAleatoires(char* input,char* inputMarqueurs,char* output,uint8_t r,uint8_t g,uint8_t b,uint16_t rayon)	Segmentação de imagens. rayon=1. Suporta imagens BMP de 24 bits.
void Watershed(char*	O algoritmo divisor de águas para

input,char* inputMarqueurs,char* output,uint8_t r,uint8_t g,uint8_t b,uint8_t a,uint16_t rayon)	segmentação de imagens. inputMarqueurs é a imagem anotada da imagem de entrada. A é geralmente 255, e rayon=1. Suporta imagens BMP de 24 bits.
double CharacterRecognition(char* TargetImage,char* TemplateFileGroup[])	Correspondência de caracteres, suporta imagens BMP, e o valor de retorno é o número de sequência do arquivo de modelo correspondente à imagem de destino. Se o valor de retorno é 2, ele indica que a imagem corresponde ao modelo com o número de sequência 2 (a partir de zero). referência : 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[])	Correspondência de caracteres, suporta imagens BMP, e o valor de retorno é o número de sequência do arquivo de modelo correspondente à imagem de destino. Se o valor de retorno é 2, ele indica que a imagem corresponde ao modelo com o número de sequência 2 (a partir de zero). referência : 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	Segmentação de caracteres. Suporta imagens BMP. OutputFolder é a pasta para a saída de resultados, como "output". O formato do nome do arquivo para a saída de resultados é: X coordenada no canto superior esquerdo - Y coordenada no canto superior esquerdo - X coordenada no canto inferior direito - Y coordenada no canto inferior direito ,

)	YHistogramValleyMaxPixelNumber é o número mínimo de pixels pretos no vale do histograma de direção Y, YHistogramValleyMaxPixelNumber=0, XHistogramValleyMaxPixelNumber é o número mínimo de pixels pretos no vale do histograma de direção X, XHistogramValleyMaxPixelNumber=4, SubImgBlackPixelPercentage é a porcentagem de pixels pretos em um subgrafo que é considerado um número , SubImgBlackPixelPercentage=0.001, SingleNumberImgBoundary é a largura de preenchimento de borda de uma única imagem digital , SingleNumberImgBoundary=5,Infinito é considerado infinito , Infinite=249480 , NumberImageBlackPixelPercentage é o número de pixels pretos em uma única imagem digital que excede todas as imagens digitais , 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)	Segmentação de caracteres. Suporta imagens BMP. BinaryGap é o limite global para binarização de imagens , BinaryGap=135, BoundaryRemoveGap é a distância em que todas as bordas estão definidas para branco , BoundaryRemoveGap=7 , Infinito é considerado infinito , Infinite=249480 , SingleNumberImgBoundary é a largura de preenchimento de borda de uma única imagem digital , SingleNumberImgBoundary=5 , YHistogramValleyMaxPixelNumber é o número mínimo de pixels pretos no vale do histograma de direção Y, YHistogramValleyMaxPixelNumber=0, XHistogramValleyMaxPixelNumber é o

	número mínimo de pixels pretos no vale do histograma de direção X, XHistogramValleyMaxPixelNumber=4, SubImgBlackPixelPercentage é a porcentagem de pixels pretos em um subgrafo que é considerado um número , SubImgBlackPixelPercentage=0.001, NumberImageBlackPixelPercentage é o número de pixels pretos em uma única imagem digital que excede todas as imagens digitais , NumberImageBlackPixelPercentage=0.35。 Referência: output="output"。
void CodeEncoding(std::string input,char* output, int width,int height, int margin, int eccLevel, int stride_bytes, int comp,int a)	Codificação de código QR. Entrada é a string a ser codificada, e saída é o nome do arquivo da imagem gerada do código QR. margem: A margem em torno do código de barras ecc: Nível de correcção de erros, [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 Referência: margin=10, eccLevel=-1, stride_bytes=0, comp=1。
std::string CodeDecoding(char* input,int req_comp,int a)	Descodificação de código QR. Entrada é o nome do arquivo da

	imagem de código QR de entrada e retorna o resultado de decodificação. a=1：Lum a=2：RGB a=3：BGR a=4：RGBX a=5：XRGB a=6：BGRX a=7：XBGR Referência：req_comp=4，a=4。