**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 dx[],int dy[],int a,double b1,int DX1,int DY1,double EPS) | Fusão de imagens. referência：a=3，b1=4，DX1=-68，DY1=-99，EPS=1，input1=" Fusão de imagens 1.jpg"，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 \*CipherFile) | Função de encriptação DES, suportando vários ficheiros. 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\* input,char\* inputMarqueurs,char\* output,uint8\_t r,uint8\_t g,uint8\_t b,uint8\_t a,uint16\_t rayon) | O algoritmo divisor de águas para 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。 |