**取扱説明書**

|  |  |
| --- | --- |
| image\_t\* ReadPNM(char\* input) | PNMファイルを読み取り、PBM、PGM、PPMイメージをサポートします。  以下の構造体を導入する必要がある：  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; //グレースケール  uint8\_t i; //カラーインデックス  } pixcel\_t;  typedef struct image\_t {  uint32\_t width; //幅  uint32\_t height; //高い  uint16\_t color\_type; //色の種類  uint16\_t palette\_num; //パレット数  color\_t \*palette; //パレットへのポインタ  pixcel\_t \*\*map; //画像データ  } image\_t; |
| void WritePNM(image\_t\* input,char\* output,int type) | PNM画像データは画像ファイルとして保存され、PBM、PGM、PPM画像をサポートします。typeは、type=1、2、3、4、5、6などのPNMファイルのフォーマットです。  以下の構造体を導入する必要がある：  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; //グレースケール  uint8\_t i; //カラーインデックス  } pixcel\_t;  typedef struct image\_t {  uint32\_t width; //幅  uint32\_t height; //高い  uint16\_t color\_type; //色の種類  uint16\_t palette\_num; //パレット数  color\_t \*palette; //パレットへのポインタ  pixcel\_t \*\*map; //画像データ  } image\_t; |
| image\_t\* ReadBMP(char\* input) | BMP画像を読み込みます。  以下の構造体を導入する必要がある：  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; //グレースケール  uint8\_t i; //カラーインデックス  } pixcel\_t;  typedef struct image\_t {  uint32\_t width; //幅  uint32\_t height; //高い  uint16\_t color\_type; //色の種類  uint16\_t palette\_num; //パレット数  color\_t \*palette; //パレットへのポインタ  pixcel\_t \*\*map; //画像データ  } image\_t; |
| void WriteBMP(image\_t\* input,char\* output,int compress) | BMP画像データは画像ファイルとして保存され、compress=1の場合はRLE圧縮される。  以下の構造体を導入する必要がある：  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; //グレースケール  uint8\_t i; //カラーインデックス  } pixcel\_t;  typedef struct image\_t {  uint32\_t width; //幅  uint32\_t height; //高い  uint16\_t color\_type; //色の種類  uint16\_t palette\_num; //パレット数  color\_t \*palette; //パレットへのポインタ  pixcel\_t \*\*map; //画像データ  } image\_t; |
| void WriteBMP(image\_t\* input,char\* output) | BMP画像データは画像ファイルとして保存される。  以下の構造体を導入する必要がある：  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; //グレースケール  uint8\_t i; //カラーインデックス  } pixcel\_t;  typedef struct image\_t {  uint32\_t width; //幅  uint32\_t height; //高い  uint16\_t color\_type; //色の種類  uint16\_t palette\_num; //パレット数  color\_t \*palette; //パレットへのポインタ  pixcel\_t \*\*map; //画像データ  } image\_t; |
| void WriteBMP1(image\_t\* input,char\* output,int compress) | BMP画像データは画像ファイルとして保存され、compress=1の場合はRLE圧縮される。  以下の構造体を導入する必要がある：  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; //グレースケール  uint8\_t i; //カラーインデックス  } pixcel\_t;  typedef struct image\_t {  uint32\_t width; //幅  uint32\_t height; //高い  uint16\_t color\_type; //色の種類  uint16\_t palette\_num; //パレット数  color\_t \*palette; //パレットへのポインタ  pixcel\_t \*\*map; //画像データ  } image\_t; |
| void ImageFusion(char\* input1,char\* input2,char\* output,int block\_height,int block\_width,double threshold) | 多焦点画像の融合、8ビットBMP画像をサポートする。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) | 画像融合。リファレンス：a=3，b1=4，DX1=-68，DY1=-99，EPS=1，input1="画像融合1.jpg"，input2="画像融合2.jpg"，MaskImage="マスク.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) | 画像融合、PNG画像をサポートする。リファレンス：input1=” 画像融合1.png”，input2=” 画像融合2.png”， inputUniqe1=” 画像融合1\_unique.txt”， inputUniqe2=” 画像融合2\_unique.txt”。 |
| void Uniqe(char\* input,char\* inputUniqe,char\* output,double R,double G,double B) | 画像融合、PNG画像をサポートする。リファレンス：input=” 画像融合1.png”， inputUniqe=” 画像融合1\_unique.txt”。R=255，G=0，B=0。 |
| void Screenshot1(HWND hWnd, LPCWSTR OutputImage) | スクリーン関数。hWndは、スクリーンショットするウィンドウハンドルです。たとえば、次のようになります：GetDesktopWindow()；OutputImageはスクリーンショット名です。 |
| void Screenshot2(HWND hWnd,LPCWSTR OutputImage) | スクリーン関数。hWndは、スクリーンショットするウィンドウハンドルです。たとえば、次のようになります：GetDesktopWindow()；OutputImageはスクリーンショット名です。 |
| void Screenshot3(HWND hWnd, LPCWSTR OutputImage) | スクリーン関数。hWndは、スクリーンショットするウィンドウハンドルです。たとえば、次のようになります：GetDesktopWindow()；OutputImageはスクリーンショット名です。 |
| uint8\_t\* AESencrypt(uint8\_t\* input,uint8\_t\* key,int size) | AES暗号化関数、inputは元データ、keyは鍵、sizeはinputのサイズです。暗号化結果データを返します。 |
| uint8\_t\* AESdecrypt(uint8\_t\* input,uint8\_t\* key,int size) | AES復号関数、inputは暗号化されたデータ、keyは鍵、sizeはinputのサイズです。復号結果データを返します。 |
| void DES\_Encrypt(char \*PlainFile, char \*Key,char \*CipherFile) | DES暗号化関数で、複数のファイルをサポートします。PlainFileは元のファイルのファイル名、Keyはキー文字、CipherFileは暗号化されたファイル名です。 |
| void DES\_Decrypt(char \*CipherFile, char \*Key,char \*PlainFile) | DES復号関数は、複数のファイルをサポートします。CipherFileは暗号化されたファイルのファイル名、Keyはキー文字、PlainFileは復号後のファイル名です。 |
| int Equal(char\* input1,char\* input2,double c) | 比画像の勾配振幅類似性偏差値がcに等しければ通過する。input 1とinput 2は比較対象の2つの画像である。cは参照の閾値である。24ビットBMP画像をサポートしています。 |
| int GreaterThan(char\* input1,char\* input2,double c) | 比画像の勾配振幅類似性偏差値がcより大きければ通過する。input 1とinput 2は比較対象の2つの画像である。cは参照の閾値である。24ビットBMP画像をサポートしています。 |
| int LessThan(char\* input1,char\* input2,double c) | 比画像の勾配振幅類似性偏差値がc未満であれば通過する。input 1とinput 2は比較対象の2つの画像である。cは参照の閾値である。24ビットBMP画像をサポートしています。 |
| double GMSD(char\* input1, char\* input2) | 2枚の画像の勾配振幅類似性偏差値を求め、結果を返す。input 1とinput 2は比較対象の2つの画像である。24ビットBMP画像をサポートしています。 |
| void FileWrite(char\* BMP,char\* TXT) | 画像が暗黙的に書かれたファイルが書き込まれ、テキストファイルが画像に書き込まれる。32ビットBMP画像をサポートしています。BMPは書き込む画像ファイル名であり、TXTは画像を書き込むテキストファイル名である。 |
| void FileWriteOut(char\* BMP,char\* TXT) | 画像を隠して書いたファイルを書き出し、テキストファイルを画像から取り出します。32ビットBMP画像をサポートしています。BMPは書き出す画像ファイル名であり、TXTは画像を書き出すと情報が保存されるテキストファイル名である。 |
| void Watershed2(char\* input,char\* inputMarqueurs,char\* output,int r,unsigned char R,unsigned char G,unsigned char B) | 画像分割の分水嶺アルゴリズム。inputMarqueursは入力画像のマーキング画像である。R=230，G=0，B=0，r=1。24ビットBMP画像をサポートしています。 |
| void EcrireImage1(char\* input,char\* output,uint32\_t rayon) | 画像分割。rayon=5。24ビットBMP画像をサポートしています。 |
| void EcrireImage2(char\* input,char\* inputMarqueurs,char\* output,uint32\_t rayon) | 画像分割。rayon=5。24ビットBMP画像をサポートしています。 |
| void EcrireLPECouleur1(char\* input,char\* inputMarqueurs,char\* output,uint32\_t rayon) | 画像分割。rayon=5。24ビットBMP画像をサポートしています。 |
| void Watershed1(char\* input,char\* inputMarqueurs,char\* output,uint32\_t rayon) | 画像分割の分水嶺アルゴリズム。inputMarqueursは入力画像のマーキング画像である。rayon=5。24ビットBMP画像をサポートしています。 |
| void EcrireImage3(char\* input,char\* inputMarqueurs,char\* output,uint16\_t rayon) | 画像分割。rayon=1。24ビットBMP画像をサポートしています。 |
| void EcrireImageCouleursAleatoires(char\* input,char\* inputMarqueurs,char\* output,uint8\_t r,uint8\_t g,uint8\_t b,uint16\_t rayon) | 画像分割。rayon=1。24ビットBMP画像をサポートしています。 |
| void Watershed(char\* input,char\* inputMarqueurs,char\* output,uint8\_t r,uint8\_t g,uint8\_t b,uint8\_t a,uint16\_t rayon) | 画像分割の分水嶺アルゴリズム。inputMarqueursは入力画像のマーキング画像である。aは一般的に255、rayon=1である。24ビットBMP画像をサポートしています。 |
| double CharacterRecognition(char\* TargetImage,char\* TemplateFileGroup[]) | 文字マッチング、BMP画像をサポートし、戻り値はターゲット画像がマッチングしたテンプレートファイルのシーケンス番号であり、戻り値が2であれば画像とシーケンス番号が2（シーケンス番号がゼロから始まる）のテンプレートのマッチングを説明する。  リファレンス：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[]) | 文字マッチング、BMP画像をサポートし、戻り値はターゲット画像がマッチングしたテンプレートファイルのシーケンス番号であり、戻り値が2であれば画像とシーケンス番号が2（シーケンス番号がゼロから始まる）のテンプレートのマッチングを説明する。  リファレンス：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) | 文字分割。BMP画像をサポートしています。  OutputFolderは結果出力のフォルダであり、「output」のように、結果を出力するファイル名の構成方法は、左上のX座標-左上のY座標-右下のX座標-右下のY座標、  YHistogramValleyMaxPixelNumberはY方向ヒストグラムを求めるので、谷の最少の黒い画素の個数，YHistogramValleyMaxPixelNumber=0，XHistogramValleyMaxPixelNumberはX方向ヒストグラムを求めるので、谷の最少の黒い画素の個数，XHistogramValleyMaxPixelNumber=4，SubImgBlackPixelPercentageは、サブマップ内の黒のピクセルが一定パーセントを超えている場合にのみ数値になります，SubImgBlackPixelPercentage=0.001，SingleNumberImgBoundaryは、1枚のデジタル画像エッジの塗り幅です，SingleNumberImgBoundary=5，Infiniteは無限大とみなす，Infinite=249480，NumberImageBlackPixelPercentageは、1枚のデジタル画像の黒画素数がすべてのデジタル画像を上回る，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) | 文字分割。BMP画像をサポートしています。  BinaryGapは画像二値化グローバル閾値である，BinaryGap=135，BoundaryRemoveGapはエッジがすべて白に設定された距離です，BoundaryRemoveGap=7，インフィニットは無限大とみなす，Infinite=249480，SingleNumberImgBoundaryは、1枚のデジタル画像エッジの塗り幅です，SingleNumberImgBoundary=5，YHistogramValleyMaxPixelNumberはY方向ヒストグラムを求めるので、谷の最少の黒い画素の個数，YHistogramValleyMaxPixelNumber=0，XHistogramValleyMaxPixelNumberはX方向ヒストグラムを求めるので、谷の最少の黒い画素の個数，XHistogramValleyMaxPixelNumber=4，SubImgBlackPixelPercentageは、サブマップ内の黒のピクセルが一定パーセントを超えている場合にのみ数値になります，SubImgBlackPixelPercentage=0.001，NumberImageBlackPixelPercentageは、1枚のデジタル画像の黒画素数がすべてのデジタル画像を上回る，NumberImageBlackPixelPercentage=0.35。  リファレンス：output="output"。 |
| void CodeEncoding(std::string input,char\* output, int width,int height, int margin, int eccLevel, int stride\_bytes, int comp,int a) | 2次元コード符号化。inputは符号化する文字列であり、outputは生成される2次元コード画像ファイル名である。  margin：バーコード周辺のマージン  ecc：誤り訂正レベル，[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  リファレンス：margin=10，eccLevel=-1，stride\_bytes=0，comp=1。 |
| std::string CodeDecoding(char\* input,int req\_comp,int a) | 2次元コード復号。inputは入力された2次元コード画像ファイル名であり、復号結果を返す。  a=1：Lum  a=2：RGB  a=3：BGR  a=4：RGBX  a=5：XRGB  a=6：BGRX  a=7：XBGR  リファレンス：req\_comp=4，a=4。 |