

## 取扱説明書

<pre>image_t* ReadPNM(char* input)</pre>	<p>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。</p> <p>以下の構造体を導入する必要がある：</p> <pre>typedef struct color_t {     uint8_t r; //Red     uint8_t g; //Green     uint8_t b; //Blue     uint8_t a; //Alpha } color_t; typedef union pixel_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 } pixel_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     pixel_t **map;         //画像データ } image_t;</pre>
<pre>void WritePNM(image_t* input, char* output, int type)</pre>	<p>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。</p> <p>以下の構造体を導入する必要がある：</p> <pre>typedef struct color_t {     uint8_t r; //Red     uint8_t g; //Green     uint8_t b; //Blue     uint8_t a; //Alpha } color_t; typedef union pixel_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 } pixel_t; typedef struct image_t {</pre>

	<pre> 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; </pre>
image_t* ReadBMP(char* input)	<p>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：</p> <pre> 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; </pre>
void WriteBMP(image_t* input, char* output, int compress)	<p>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：</p> <pre> typedef struct color_t {     uint8_t r; //Red     uint8_t g; //Green     uint8_t b; //Blue     uint8_t a; //Alpha } color_t; </pre>

	<pre> typedef union pixel_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 } pixel_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     pixel_t **map;         //画像デー                              タ } image_t; </pre>
<pre> void      WriteBMP(image_t* input, char* output) </pre>	<p>BMP 画像データは画像ファイルとして保存される。</p> <p>以下の構造体を導入する必要がある：</p> <pre> typedef struct color_t {     uint8_t r; //Red     uint8_t g; //Green     uint8_t b; //Blue     uint8_t a; //Alpha } color_t;  typedef union pixel_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 } pixel_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     pixel_t **map;         //画像デー                              タ } image_t; </pre>
<pre> void      WriteBMP1(image_t* input, char*      output, int compress) </pre>	<p>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。</p>

	<p>以下の構造体を導入する必要がある:</p> <pre> typedef struct color_t {     uint8_t r; //Red     uint8_t g; //Green     uint8_t b; //Blue     uint8_t a; //Alpha } color_t; typedef union pixel_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 } pixel_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     pixel_t **map;         //画像データ } image_t; </pre>
<pre> void ImageFusion(char* input1, char*      input2, char* output, int        block_height, int block_width, double threshold) </pre>	<p>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。</p>
<pre> void ImageFusion(char* input1, char*      input2, char* MaskImage, char*   output, int dx[], int dy[], int a, double b1, int DX1, int DY1, double EPS) </pre>	<p>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".</p> <pre> int dx[] = {0, 0, -1, 1}; int dy[] = {-1, 1, 0, 0}; </pre>
<pre> void ImageFusion(char* input1, char*      input2, char* inputUnique1, char* inputUnique2, char* output) </pre>	<p>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" , inputUnique1=" 画像融合 1_unique. txt" , inputUnique2=" 画像融合 2_unique. txt" 。</p>
<pre> void Unique(char* input, char* inputUnique, char* output, double R, double G, double B) </pre>	<p>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" , inputUnique=" 画像融合 1_unique. txt" 。R=255, G=0, B=0。</p>

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*	画像分割。rayon=1。24 ビット BMP 画像をサポートしています。

output, uint8_t r, uint8_t g, uint8_t b, uint16_t rayon)	
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

	<p>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。</p>
<pre>void CharacterSegmentation(char* input, char*      output,      int BoundaryRemoveGap,      int BinaryGap, int YHistogramValleyMaxPixelNumber, double SubImgBlackPixelPercentage, int Infinite,      int XHistogramValleyMaxPixelNumber, double NumberImageBlackPixelPercentage , int SingleNumberImgBoundary)</pre>	<p>文字分割。BMP 画像をサポートしています。</p> <p>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，</p>



	<p>NumberImageBlackPixelPercentage=0.35。</p> <p>リファレンス: output="output"。</p>
<pre>void CodeEncoding(std::string input, char* output, int width, int height, int margin, int eccLevel, int stride_bytes, int comp, int a)</pre>	<p>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。</p> <p>margin: バーコード周辺のマージン</p> <p>ecc: 誤り訂正レベル, [0-8]</p> <p>a=1: AZTEC</p> <p>a=2: CODABAR</p> <p>a=3: CODE_39</p> <p>a=4: CODE_93</p> <p>a=5: CODE_128</p> <p>a=6: DATA_MATRIX</p> <p>a=7: EAN_8</p> <p>a=8: EAN_13</p> <p>a=9: ITF</p> <p>a=10: MAXICODE</p> <p>a=11: PDF_417</p> <p>a=12: QR_CODE</p> <p>a=13: RSS_14</p> <p>a=14: RSS_EXPANDED</p> <p>a=15: UPC_A</p> <p>a=16: UPC_E</p> <p>a=17: UPC_EAN_EXTENSION</p> <p>リファレンス: margin=10, eccLevel=-1, stride_bytes=0, comp=1。</p>
<pre>std::string CodeDecoding(char* input, int req_comp, int a)</pre>	<p>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。</p> <p>a=1: Lum</p> <p>a=2: RGB</p> <p>a=3: BGR</p> <p>a=4: RGBX</p> <p>a=5: XRGB</p> <p>a=6: BGRX</p> <p>a=7: XBGR</p> <p>リファレンス: req_comp=4, a=4。</p>