ディジタル映像処理及び演習

第 1 回

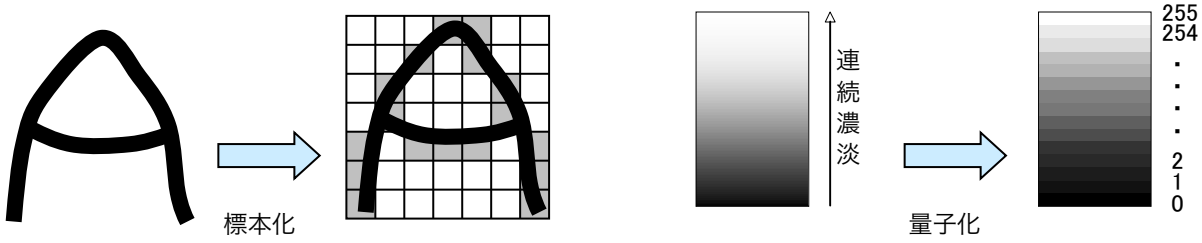
ディジタル画像の基礎知識

○ディジタル画像の構造

ディジタル画像は一般的に，連続的な座標と輝度値を持つアナログ画像を**標本化**および**量子化**することによって生成される．

ディジタル画像生成の**標本化**とは，連続的な座標値に二次元格子を当てはめて離散的な座標値に変換することである．格子数によってディジタル画像の**画像サイズ**(幅と高さの画素数)が決まり，各格子がディジタル画像の画素となって輝度値を持つことになる．

ディジタル画像生成の**量子化**とは，各画素の連続的な輝度値を離散的な値に変換することである．輝度値をどのくらい細かく分割するか，つまり階調を決めるのが画素値の**デプス**であり，一般的には 8 ビットデプス(0〜255：256 階調)が使われることが多い．



○グレースケール画像とカラー画像

ディジタル画像には大きく分けて**グレースケール画像**と**カラー画像**がある．**グレースケール画像**は各画素が画像の明るさを示す 1 つの値，つまり **1 チャンネル**だけを持つ画像である．一方，**カラー画像**は色情報を持つ画像で，最もよく使われるカラー画像は各画素が R(赤)，G(緑)，B(青)の 3 つの値，つまり **3 チャンネル**を持つ．4 チャンネル持つカラー画像もある．

カラー画像の表現方法は様々で，**RGB 形式**の他，**HSV 形式**，**XYZ 形式**，**Lab 形式**，**CMYK 形式**などがあり，用途に応じて使い分けられている．カラー画像の色の表現形式を表色系と呼ぶ．

カラー画像から明るさ情報だけを取り出すことで 1 チャンネルのグレースケール画像を生成することができる．カラー画像からグレースケール画像を生成する方法には様々な種類がある．例えば，カラー画像の G チャンネルだけを使用すれば簡易的なグレースケール画像が生成できる．実際には，RGB の各チャンネルを適切な割合で足し合わせたり，HSV カラー画像の V 成分を用いたりして，適切なグレースケール画像を生成する．

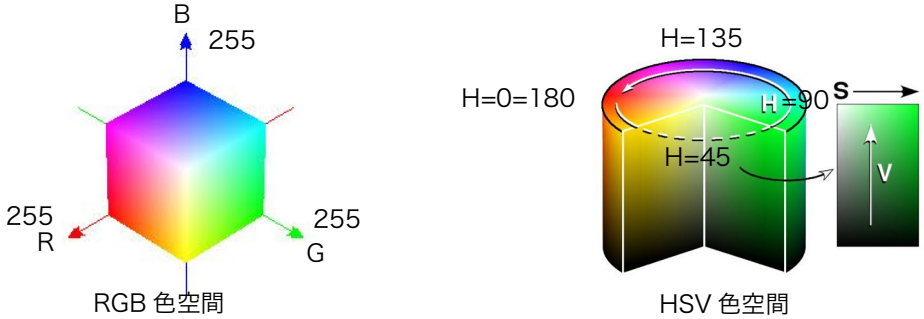
グレースケール画像：1 チャンネル　　カラー画像：3 チャンネル

○RGB カラー画像と HSV(HSI)カラー画像

RGB は色を光の 3 原色の **R(赤)**，**G(緑)**，**B(青)**の 3 チャンネルで表現する色空間である．OpenCV のデプスが 8 ビットの RGB 画像では，RGB のそれぞれの値が 0〜255 の 256 段階の値を取り，256^3，つまり 1677 万 7216 色を表現することができる．

HSV は色を **H(色相)**，**S(彩度)**，**V(明度)**の 3 チャンネルで表現する色空間である． H(色相)は色の違いを表す属性で，赤，黄，緑，青，紫などの色の種類を表現する．S(彩度)は色の鮮やかさを表す属性で，同じ色相で無彩色グレーから鮮やかな色を表現する．V(明度)は色の明るさを表す属性である．V チャンネルだけを取り出せば適切なグレースケール画像を生成することができる．OpenCV の HSV カラー画像では，H は 0〜180，S と V が 0〜255 の値を取る．

OpenCV では，RGB カラー画像は B(青)，G(緑)，赤(赤)の順番で色が格納されている．また，RGB カラー画像⇔HSV カラー画像の変換を行う関数が用意されている（`cv::cvtColor()`）．



画像ファイルの読み込みと簡単な処理

○DIP01 のソースコード（"dip01.cpp"）

```
#include <iostream>  //入出力関連ヘッダ
#include <opencv2/opencv.hpp>  //OpenCV 関連ヘッダ

int main (int argc, const char* argv[])
{
    //①画像ファイルの読み込み
    cv::Mat sourceImage = cv::imread("color.jpg", cv::IMREAD_COLOR);
    if (sourceImage.data==0) {  //画像ファイルが読み込めなかった場合
        printf("File not found\n");
        exit(0);
    }
    printf("Width=%d, Height=%d\n", sourceImage.cols, sourceImage.rows);  //横幅と高さ

    //②画像格納用インスタンスの生成
    cv::Mat grayImage;  //cv::Mat クラス
    std::vector<cv::Mat> bgrImage(3);  //cv::Mat クラスの動的配列(初期要素数 3)

    //③ウィンドウの生成と移動
    cv::namedWindow("Source");
    cv::moveWindow("Source", 0,0);
    cv::namedWindow("Gray");
    cv::moveWindow("Gray", 400,0);
    cv::namedWindow("B");
    cv::moveWindow("B", 400,150);
    cv::namedWindow("G");
    cv::moveWindow("G", 400,300);
    cv::namedWindow("R");
    cv::moveWindow("R", 400,450);

    //④画像処理
    cv::cvtColor(sourceImage, grayImage, cv::COLOR_BGR2GRAY);
    cv::split(sourceImage, bgrImage);

    //⑤ウィンドウへの画像の表示
    cv::imshow("Source", sourceImage);
    cv::imshow("Gray", grayImage);
    cv::imshow("B", bgrImage[0]);
    cv::imshow("G", bgrImage[1]);
    cv::imshow("R", bgrImage[2]);

    //⑥キー入力待ち
    cv::waitKey(0);

    //⑦画像の保存
    cv::imwrite("gray.jpg", grayImage);

    return 0;
}
```

○プログラムのコンパイル

ターミナルで以下のコマンドを入力することで，プログラムのコンパイルが可能である．

g++ dip01.cpp -std=c++11 `pkg-config --cflags --libs opencv4`

（OpenCV3 の場合は **g++ dip01.cpp `pkg-config --cflags --libs opencv`**）

コンパイルが成功すれば，"a.out"という名前の実行ファイルが生成される．

○プログラム説明

DIP01 では，クラスやインスタンスの概念など，C++言語の形式を用いている．

・ヘッダファイル

#include <iostream>
iostream は C++言語での入出力関連ヘッダファイル．C での stdio.h のようなもの．

#include <opencv2/opencv.hpp>
OpenCV 関連のヘッダファイルをインクルード．

・手順①：画像ファイルの読み込み

OpenCV において，画像データは **cv::Mat クラス**で取り扱う．cv::Mat クラスでは，画像サイズ(幅と高さ)，デプス，チャンネル数，各画素の値に関する情報などを格納することができる．

画像ファイルは `cv::imread()`関数で読み込むことができる．

`cv::Mat` クラス

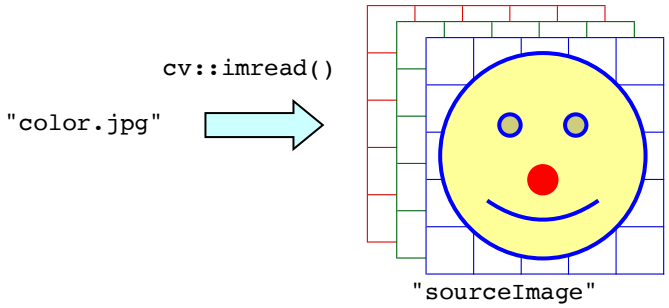
OpenCV の画像データを管理するクラス．主なメンバは以下の通り．

```
int  cols;            //画像の幅(画素数)
int  rows;            //画像の高さ(画素数)
int  channels();      //画像のチャンネル数
int  depth();         //画像のデプス識別子(CV_8U(=0)，CV_32F(=5)など)
Size  size();         //画像のサイズ
uchar *data;          //画像データへのポインタ
```

`Mat cv::imread(const string& filename, int flags);`

画像ファイルを読み込んで `cv::Mat` クラスのインスタンスに格納する関数．引数 `filename` は画像ファイル名．引数 `flags` は読み込みモードで，`cv::IMREAD_UNCHANGED`（8 ビット），`cv::IMREAD_GRAYSCALE`（8 ビット 1 チャンネル），`cv::IMREAD_COLOR`（8 ビット 3 チャンネル）などを指定．

サンプルプログラムでは，画像ファイル"color.jpg"をカラー画像（8 ビット 3 チャンネル）として読み込んで cv::Mat クラスのインスタンス "sourceImage"に格納している．画像ファイルが読み込めなかった場合はメッセージを表示して終了する．

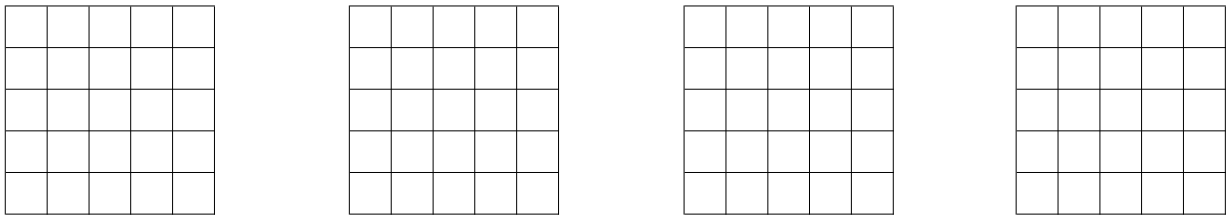


画像データに関する情報は cv::Mat クラスのインスタンスの各メンバにアクセスすることで取得することができる．サンプルプログラムでは，読み込んだ画像の幅と高さの情報にアクセスして表示している．

・手順②：画像データ用インスタンスの生成

画像処理結果を格納するなど，新たな画像データを扱うにはあらかじめ画像データを格納する cv::Mat クラスのインスタンスを生成しておく．また，複数の画像データを格納するために動的配列クラスを用いた cv::Mat クラスのインスタンスを生成することもできる．このとき，格納する画像のサイズやチャンネル数を決めておかなくても自動的に対応してくれる．

サンプルプログラムでは，1 枚の画像を格納する"`grayImage`"と，3 枚の画像を格納する"`bgrImage[]`"を生成している．



・手順③：ウィンドウの生成と移動

cv::Mat クラスの画像データはウィンドウに表示することができる．そのため，あらかじめウィンドウを生成して移動しておく．

`void cv::namedWindow(const string& name, int flags)`

ウィンドウを生成する関数．引数 `name` はウィンドウの識別に用いられるウィンドウ名で，タイトルバーにも表示．引数 `flags` はウィンドウのフラグで，通常は 省略，または `CV_WINDOW_AUTOSIZE` と記述．

`void cv::moveWindow(const string& name, int x, int y)`

ウィンドウを移動させる関数．引数 `name` はウィンドウ名．引数 `x,y` はウィンドウ左上隅の x 座標と y 座標．

サンプルプログラムでは，5 つのウィンドウを生成している．

・手順④：画像処理

OpenCV では，ある表色系の画像データを別の表色系に変換する `cv::cvtColor()`関数や，複数チャンネルを持つ画像を各チャンネルに分割する `cv::split()`関数など，様々な画像処理用関数が用意されている．

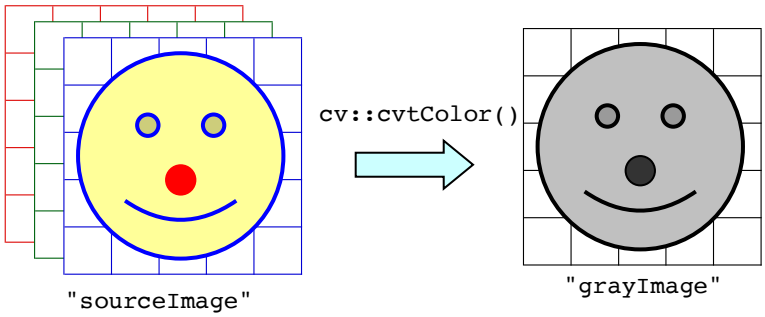
void cv::cvtColor(const Mat& src, Mat& dst, int code)
入力画像の色空間を変換する関数．引数 src は入力画像，引数 dst は変換後の出力画像をそれぞれ指定．引数 code は変換の種類を決定する定数．RGB 形式の BGR カラー画像をグレースケール画像に変換するには cv::COLOR_BGR2GRAY を指定．その他，変換種類を示す定数は以下の通り．

- cv::COLOR_BGR2GRAY：BGR カラー画像→グレースケール画像（逆変換は cv::COLOR_GRAY2BGR）
- cv::COLOR_BGR2HSV：BGR カラー画像→HSV カラー画像（逆変換は cv::COLOR_HSV2BGR）
- cv::COLOR_BGR2BGRA：BGR カラー画像→αチャンネル(不透明度)追加（逆変換は cv::COLOR_BGRA2BGR）

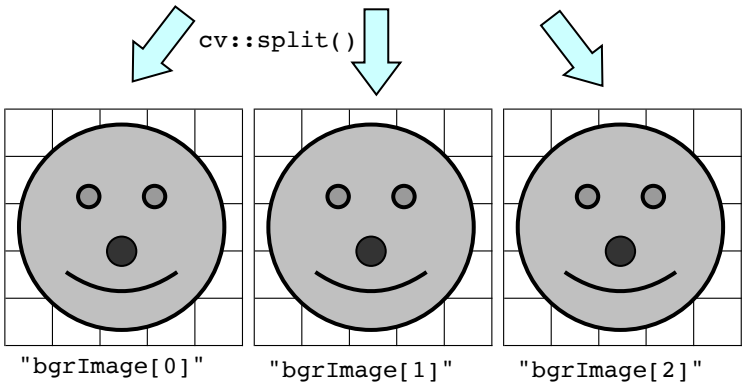
void cv::split(const Mat& src, vector<Mat>& dst)
複数チャンネルを持つ入力画像を 1 チャンネルの複数画像に分解する関数．引数 src は入力画像，引数 dst は vector 動的配列による分解後の出力画像．

サンプルプログラムでは， カラー画像"sourceImage"を cv::cvtColor()関数でグレースケール画像に変換して"grayImage"に格納するとともに，cv::split()関数でおよび B(青)，G(緑)，R(赤)のチャンネルに分解して，"bgrImage[]"に格納している．このとき，B(青)，G(緑)，R(赤)の各チャンネルはそれぞれ，"bgrImage[0]"，"bgrImage[1]"，"bgrImage[2]"に格納されている．

```
cv::cvtColor(sourceImage, grayImage, cv::COLOR_BGR2GRAY);
```




```
cv::split(sourceImage, bgrImage);
```



・手順⑤：画像の表示

cv::Mat クラスの画像データはウィンドウに表示することができる．

void cv::imshow(const string& name, const Mat& image)
ウィンドウに画像を表示する関数．引数 name は表示先のウィンドウ名を指定．引数 image は表示する画像データ．

サンプルプログラムでは，5 つの画像データをそれぞれ異なるウィンドウに表示している．

・手順⑥：キー入力待ち

OpenCV ではキー入力やマウス入力に関する関数も用意されている．

int cv::waitKey(int delay)
キーが押されるまで待機する関数．引数 delay は入力を待つ時間(ミリ秒)で，0 の場合はキー入力を無限待機．

サンプルプログラムでは，キー入力があるまで待機している．

・手順⑦：画像の保存

OpenCV では cv::Mat クラスの画像データを様々な形式の画像ファイルとして保存することができる．

bool cv::imwrite(const string& filename, const Mat& image)
画像データを画像ファイルとして保存する関数．引数 filename は保存ファイル名．ファイル名の拡張子を指定することで，様々な形式で画像を保存することが可能(".jpg"，".bmp"，".tiff"，".png")．引数 image は保存する画像データ．

サンプルプログラムでは，"gray.jpg"(JPEG 方式)というファイル名で画像データ"grayImage"を保存している．

○グレースケール画像の 2 値化しきい値処理

一般的な 8 ビットデプスグレースケール画像は各画素値が 0～255 の値を持つ．値が 0 の画素は黒色，255 の画素は白色で，その間の値はグレーとなる．

グレースケール画像の **2 値化**とは，0～255 の 256 段階の値を取る画素値を 2 つの値（0 か 255，0 か 1 など）のいずれかに変換する処理である．特にある値を**しきい値**として設定して，画素値がしきい値より低い場合は 0，しきい値より高い場合は 255（または 1）に変換する処理を **2 値化しきい値処理**と呼ぶ．2 値化しきい値処理は，画像中からの対象領域の切り出しやマスク処理などにも用いられる．

OpenCV では，しきい値処理を行うために cv::theshold()関数が用意されている．

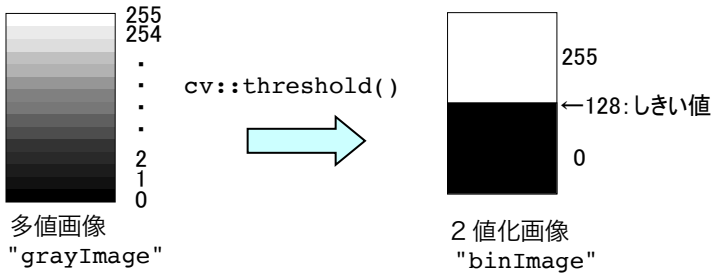
double cv::threshold(const Mat& src, Mat& dst, double thresh, double maxval, int threshtype)

グレースケール画像をしきい値処理する関数．引数 src は入力画像(1 チャンネル)，引数 dst はしきい値処理された出力画像(1 チャンネル)．引数 thresh はしきい値，引数 maxval は出力画像の画素値の最大値，引数 threshtype はしきい値処理の種類を示す定数．threshtype は以下の通り．

- cv::THRESH_BINARY：画素値が thresh より大きい場合は maxval，それ以外は 0.
- cv::THRESH_BINARY_INV：画素値が thresh より大きい場合は 0，それ以外は maxval.
- cv::THRESH_TRUNC：画素値が thresh より大きい場合は thresh，それ以外は変更なし.
- cv::THRESH_TOZERO：画素値が thresh より大きい場合は変更なし，それ以外は 0.
- cv::THRESH_TOZERO_INV：画素値が thresh より大きい場合は 0，それ以外は変更なし.

例えば，グレースケール画像 "grayImage"をしきい値 128 で 2 値化処理して "binImage" に出力する場合，引数は，thresh：**128**，maxval：**255**，threshtype：**cv::THRESH_BINARY** とすればよい．生成された 2 値化しきい値処理画像は 0 か 255 の値しか持たないグレースケール画像となる．

```
cv::threshold(grayImage, binImage, 128, 255, cv::THRESH_BINARY);
```



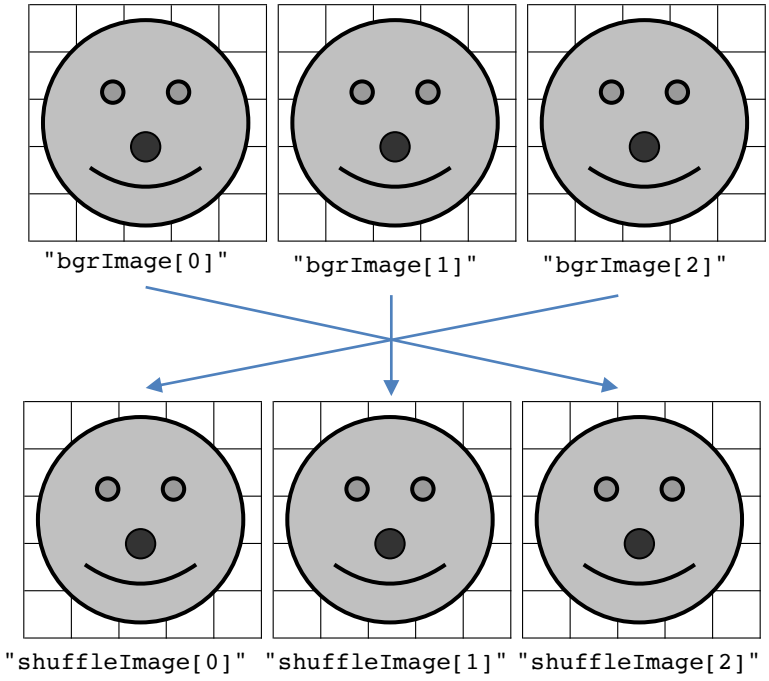
○補足：1 チャンネルの複数画像から複数チャンネル（カラー）画像の作成

複数チャンネルを持つ入力画像を 1 チャンネルの複数画像に分解する関数として cv::split()関数が用意されているが，逆に 1 チャンネルの複数画像から複数チャンネル（カラー）画像を作成する関数として cv::merge()関数も用意されている．

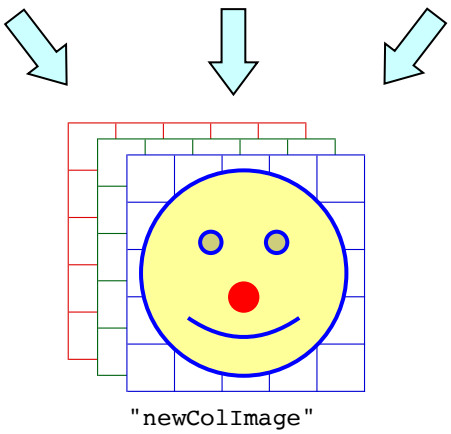
```
・・・・・・・・・・・・・・・・・・・・
int main (int argc, const char* argv[])
{
    ・・・・・・・・・・・・・・・・・・・・
    //④画像処理
    cv::cvtColor(sourceImage, grayImage, cv::COLOR_BGR2GRAY);
    cv::split(sourceImage, bgrImage);

    std::vector<cv::Mat> shuffleImage;  //チャンネル入れ替え作業用画像
    cv::Mat newColImage;  //チャンネル入れ替え結果画像

    //チャンネルの順番を入れ替えて shaffleImage を生成（shuffleImage の配列に順次画像を追加）
    shuffleImage.push_back(bgrImage[2]);
    shuffleImage.push_back(bgrImage[1]);
    shuffleImage.push_back(bgrImage[0]);

    //1 チャンネルの複数画像（shuffleImage[]）から複数チャンネル（カラー画像）を作成
    cv::merge(shuffleImage, newColImage);

    //⑤ウィンドウへの画像の表示
    ・・・・・・・・・・・・・・・・・・・・
    cv::imshow("newCol", newColImage);

    ・・・・・・・・・・・・・・・・・・・・
}
```

void cv::merge(const vector<Mat>& src, Mat& dst)

複数の 1 チャンネル入力画像を結合して 1 枚の複数チャンネル画像を生成する関数．引数 src は入力画像の配列で，引数 dst は結合によって生成された出力画像．




```
cv::merge(shuffleImage, newColImage);
```



演習課題

課題 1.

自分のカラー顔写真を原画像として, これをグレースケールに変換した画像, および2値化しきい値処理した画像を生成しなさい.

原画像カラー画像，グレースケール画像，2 値化しきい値処理画像をそれぞれウィンドウで表示した状態でスクリーンショットを撮って，1 枚の画像として提出しなさい.

＜手順＞

1. Mac 付属のソフト「Photo Booth」で自分の顔写真を撮影する．撮影した写真画像ファイルはホームディレクトリ以下の「ピクチャ」→「Photo Booth」フォルダ内に保存される．写真画像ファイル名を "photo.jpg" 等に変更して，プロジェクトフォルダ内に移動しておく.
2. ファイル読み込み部(`cv::imwrite()`関数)を "photo.jpg" を読み込むように修正する.
3. 2 値化しきい値処理した画像データを格納する cv::Mat クラスのインスタンス "`binImage`" を生成しておく.
4. `cv::thrshold()`関数を用いてグレースケール画像データ "`grayImage`" を 2 値化しきい値処理して，画像データ "`binImage`" に出力する.
5. 2 値化画像データ "`binImage`" 用のウィンドウを生成するとともに，画像を表示する.

＜処理の流れ＞

"sourceImage" "grayImage" "binImage"

課題 2.

カラー画像ファイル "kadai01-2.png"にはある「人物」の顔画像が埋め込まれている．この画像にグレースケール化と 2 値化しきい値処理を施すことで，この「人物」の顔画像（輪郭線画像）をできるかぎりはっきりと抽出しなさい.

原画像と 2 値化しきい値処理画像をそれぞれウィンドウで表示した状態でスクリーンショットを撮って，1 枚の画像として提出しなさい.

＜ヒント＞

読み込んだ画像を `cv::cvtColor()`関数を利用して”単純”にグレースケール化して，そのあと `cv::threshold()`関数でしきい値処理をしても，目的の画像が得られる可能性は低い．授業内では”別の”グレースケール化も行っているので，これを利用するといい結果が得られるかもしれない．また，`cv::threshold()`関数では適切なしきい値を見つけ出す必要がある.