画像処理及び演習

～二値画像処理 (2)～

塚田・森本・澤野

目次

- 第９回：二値画像処理(2)

✦復習（二値化処理）

✦輪郭追跡

1. 輪郭とは
2. 画素の連結性
3. 輪郭追跡の原理
4. 輪郭の記述
5. 輪郭追跡プログラム

✦領域特徴量

6. 領域特徴量

✦課題

復習

二値化処理

・画像の画素値を二つの値だけにする処理

- 各画素を明るい画素と暗い画素のどちらかに分類

➡白（255）もしくは黒（0）に変換

カラー画像

濃淡画像

二値画像

復習

二値化処理の手法

- 濃淡画像（輝度画像）から二値画像を作成
  - 固定しきい値法
    - 定まった値（しきい値）で分ける
  - pタイル法
    - 画素数の比率（面積）で分ける
  - 判別分析法
    - 輝度値の分布（分布の谷）を見て分ける

復習

しきい値処理関数

- 関数紹介

```
cv::threshold(入力画像, 出力画像, しきい値, max_value, オプション);
```

- オプション

➡ cv::THRESH_BINARY: しきい値以上をmax_valueに設定

➡ cv::THRESH_BINARY_INV: しきい値以下をmax_valueに設定

➡ cv::THRESH_OTSU：（大津の）判別分析法
（上のオプションと一緒に用いる）

- **例**

```
//しきい値処理．しきい値以上を255にする
cv::threshold(src_img, dst_img, THRESHOLD, 255, cv::THRESH_BINARY);
```

二値画像処理

・二値画像に対する処理

- 画像から位置・形状の情報を取り出すことができる

・二値画像処理の種類

✓ 輪郭追跡

✓ 領域特徴量抽出

✓ 膨張収縮処理

✓ ラベリング処理

ノート

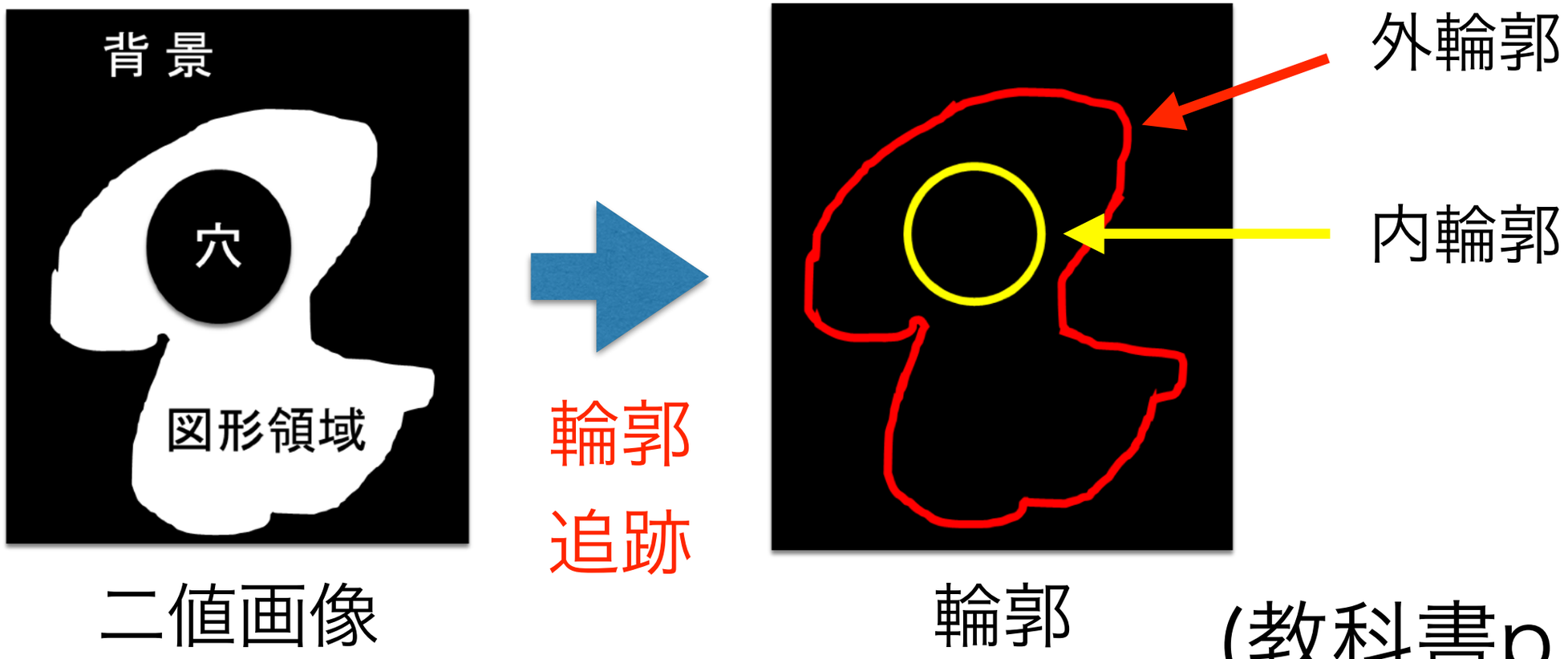
輪郭（contour）

・画素の集まりからなる図形領域を囲む特徴

✓ 図形領域と背景の境界：外輪郭

✓ 図形領域中の穴を囲む境界：内輪郭

✳　どちらも図形領域内の画素をつなげた縁（へり）



(教科書p.74)

ノート

輪郭追跡

・輪郭を求める目的

➡ 図形領域の位置・形状がわかる

➡ 図形領域の特徴がわかる（領域特徴量）

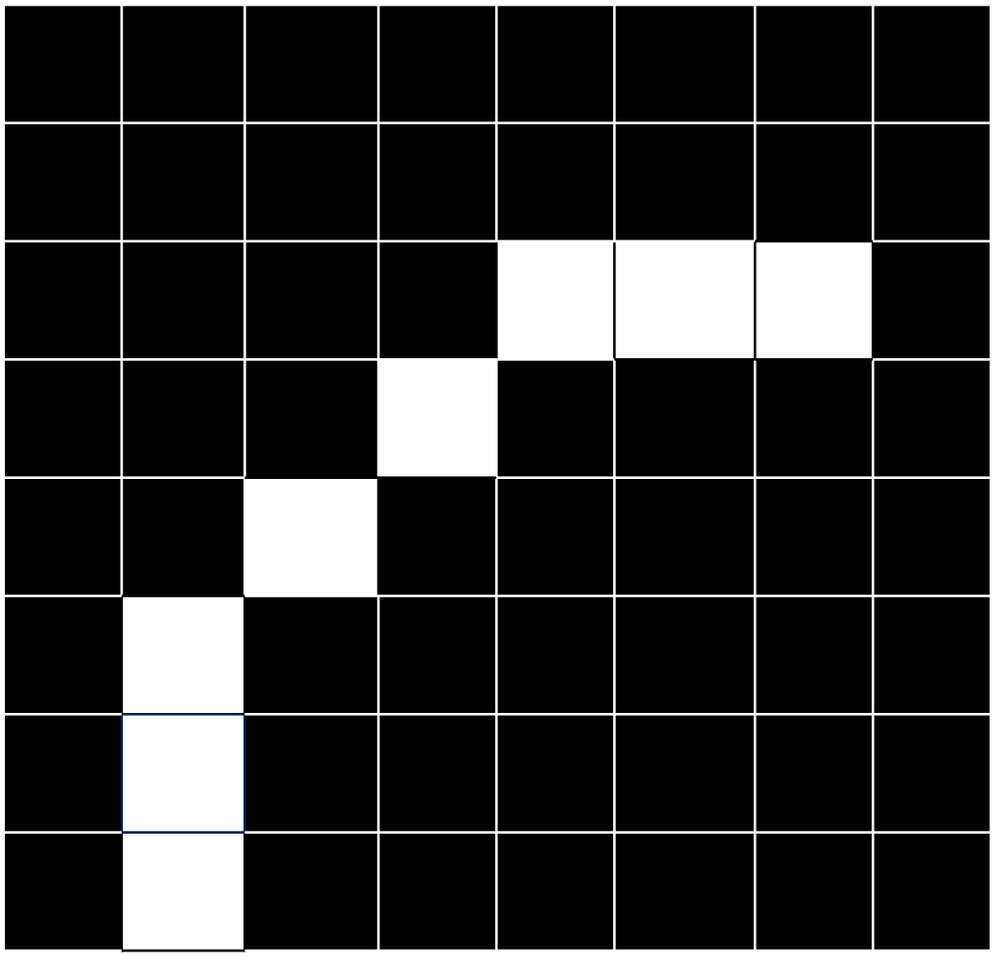
➡ 図形領域を変形できる・数えることができる　等

画像内の図形領域ごとに処理を行うことができる

・輪郭追跡

- 輪郭を構成する画素列を求める処理
- 図形領域の縁（へり）にある画素をつないでいく

この線はつながっている？



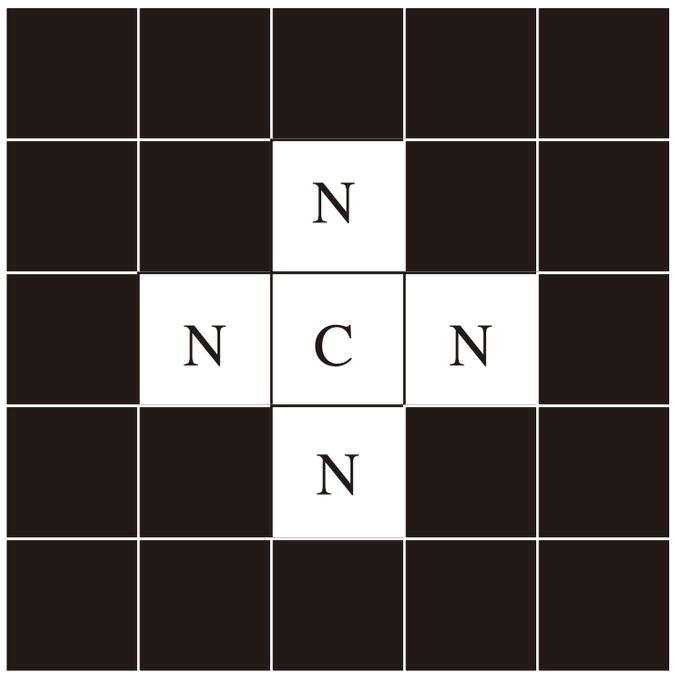
連結性によって変わる→輪郭追跡で重要

ノート

連結性

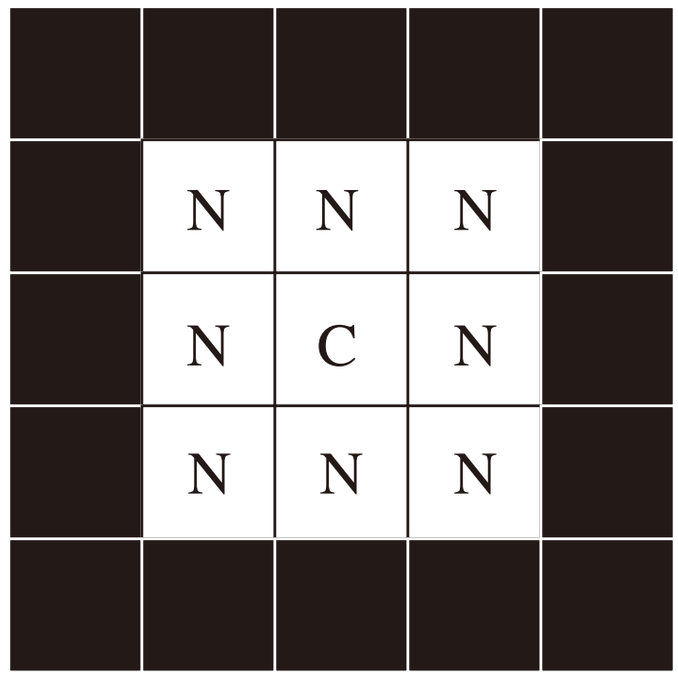
・つながっている画素：近傍（neighbor）

- 注目画素Cに対し、斜めの画素を近傍とみなすか？
（近傍の基準をどのように定めるか）→連結性



４連結：斜めは近傍でない

縦横の画素N：４近傍

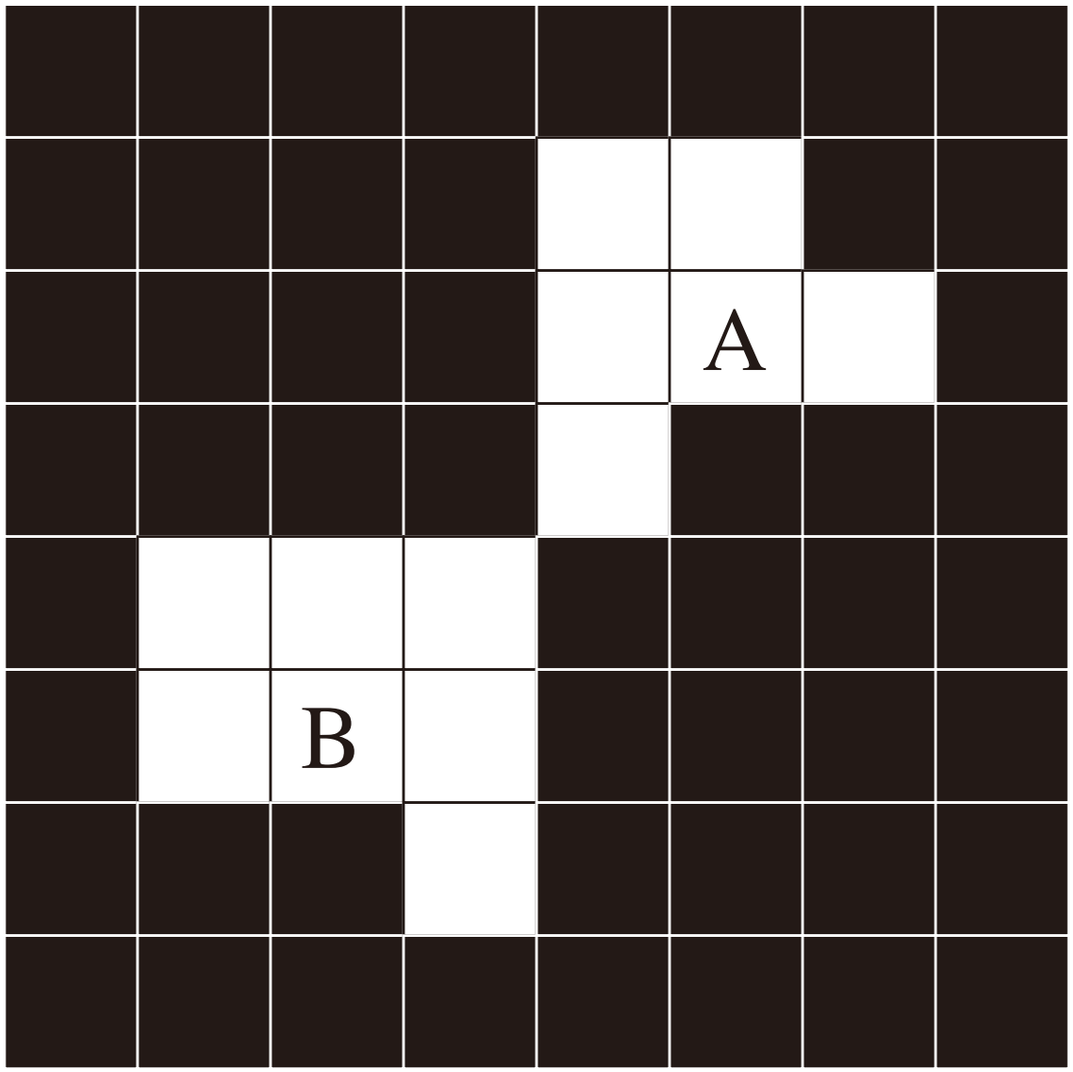


８連結：斜めは近傍である

周囲の画素N：８近傍

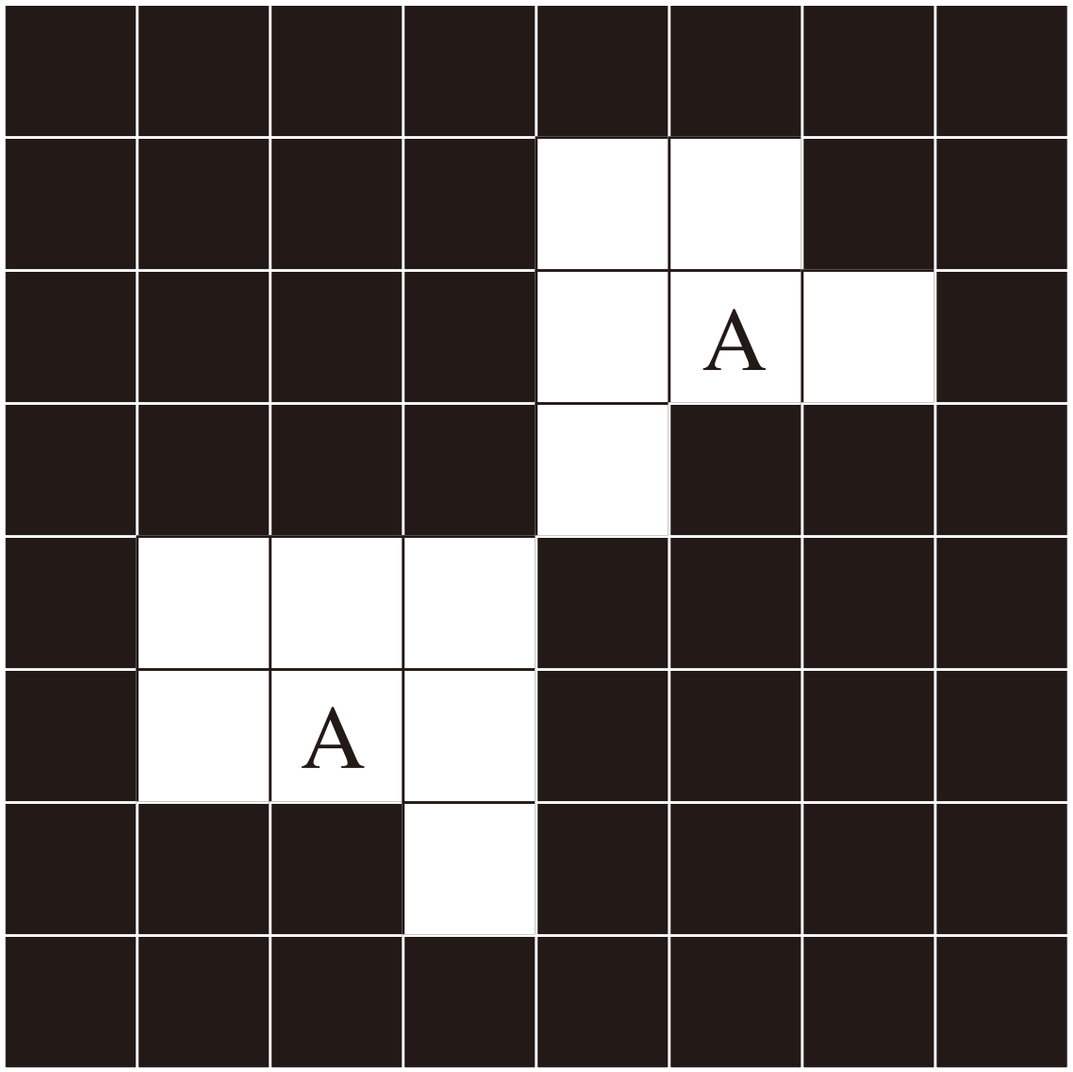
(教科書p.64)

連結性により図形領域は変わる



4連結

二つの領域となる



8連結

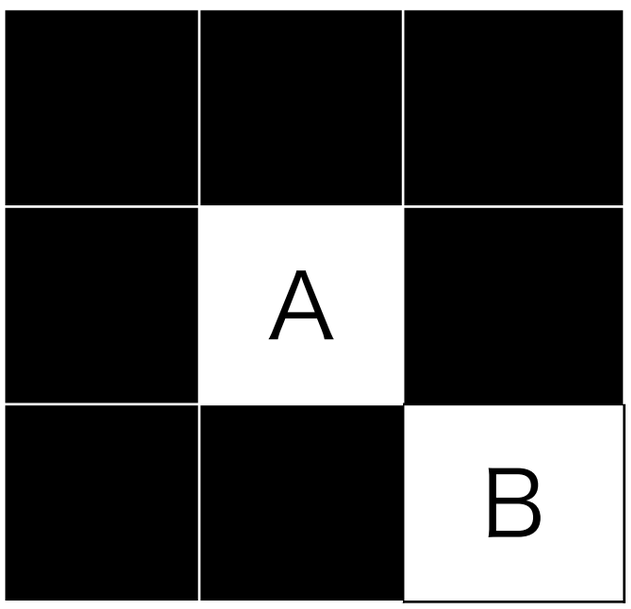
一つの領域となる

ノート

連結性と距離

- A点(i, j)とB点(k, h)の**距離**
  - ユークリッド距離　$\sqrt{(i-k)^2+(j-h)^2}$
  - 4近傍距離　$|i-k|+|j-h|$
  - 8近傍距離　$\max(|i-k|,|j-h|)$

例



ユークリッド距離：$\sqrt{2}$

4近傍距離：2

8近傍距離：1

輪郭追跡処理のポイント

(教科書p.74)

1. どの画素から処理を始めるか
2. どのように輪郭（へりの画素）を追跡するか
3. どうなったら処理を終了するか

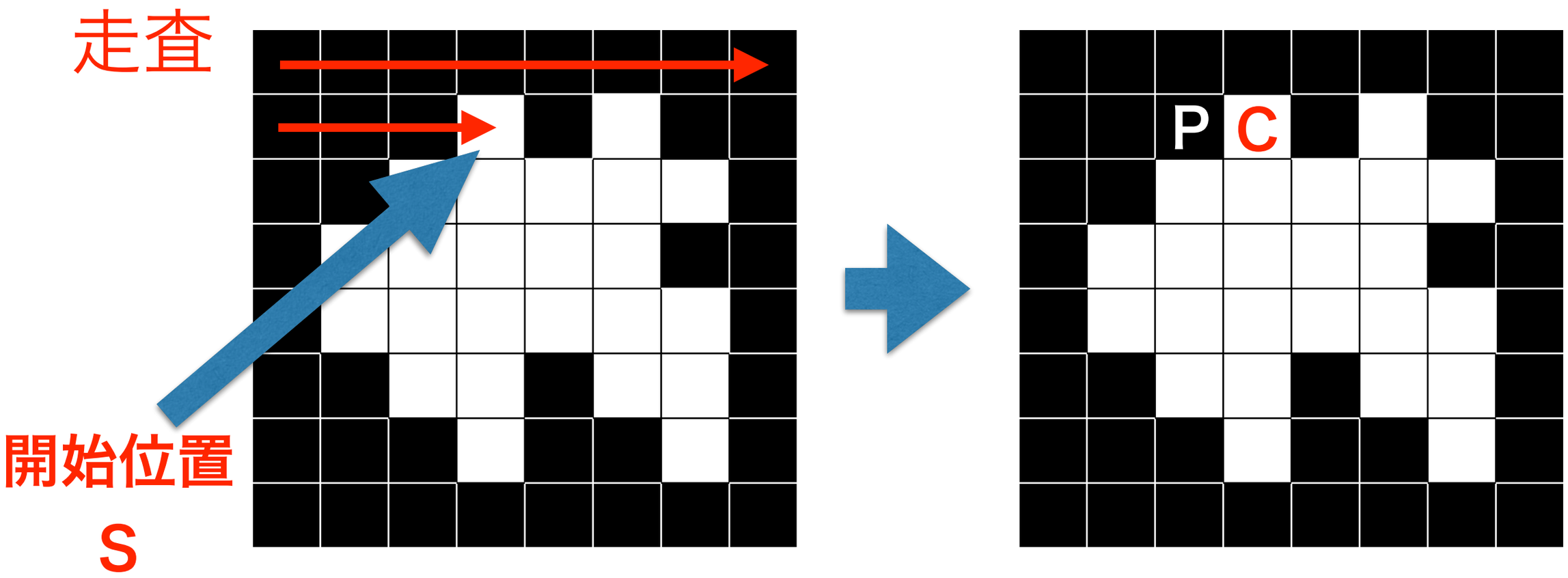
1. 画像を走査して、まだ輪郭追跡していない画素を見つける
2. 注目画素をその画素と「つながっている画素」に次々更新
3. 開始位置に戻った場合（又は2.で画素が見つからない場合）

追跡アルゴリズムその１（８連結）

1. 開始位置Sを決める

✓画像を左上から走査し、見つかった未追跡（未探索）の白画素の位置を開始位置Sとする

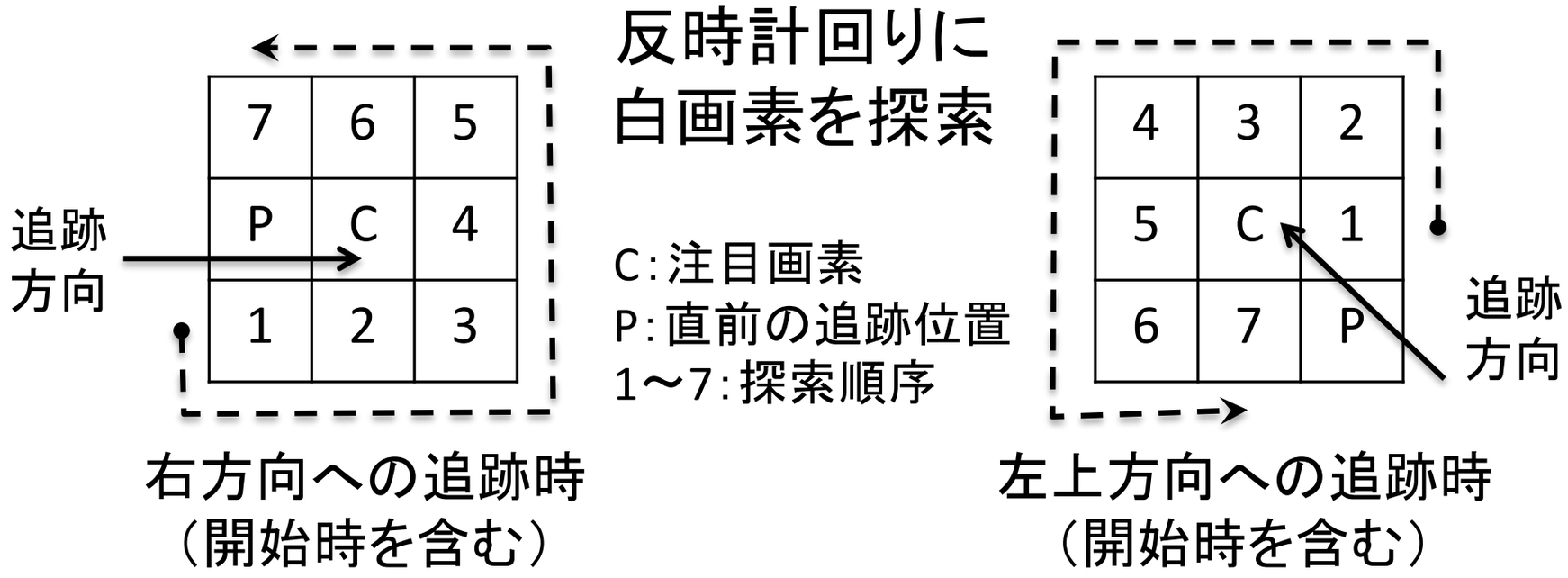
✓開始画素Sを初期注目画素C、その左隣を直前位置P



追跡アルゴリズムその２（８連結）

2. 連結画素を探索

✓注目画素Cを中心に、直前位置Pから反時計回りに白画素を探す



✓見つかった場合、その白画素を追跡済とマークし新たな注目画素Cとして更新（直前位置Pも更新）→探索続行

(教科書p.75)

追跡アルゴリズムその３（８連結）

3. 終了判定

✓2.で発見された白画素の位置が開始位置Sの場合

➡その開始位置Sでもう一度2.の白画素探索を行う

・見つかった白画素が追跡済　→　終了

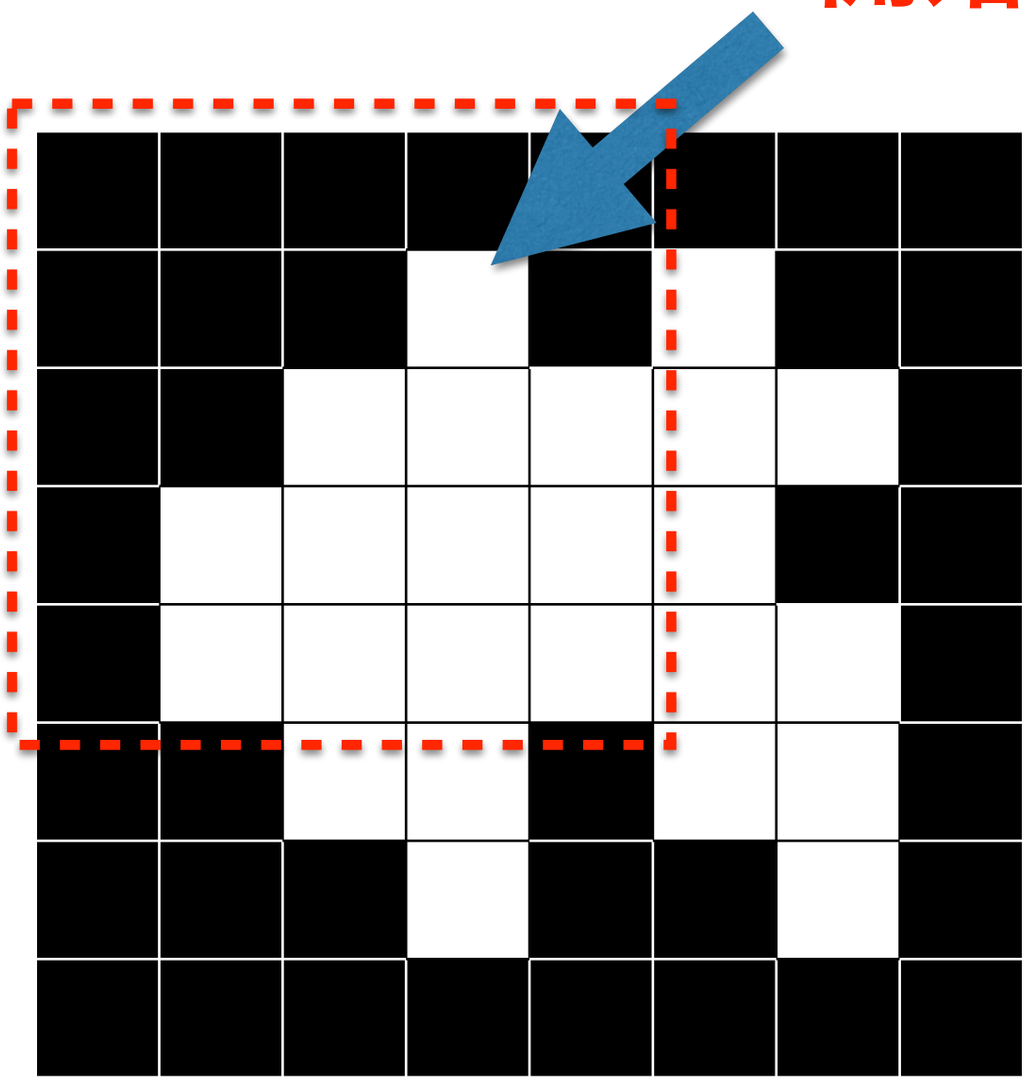
・見つかった白画素が未追跡　→　追跡続行

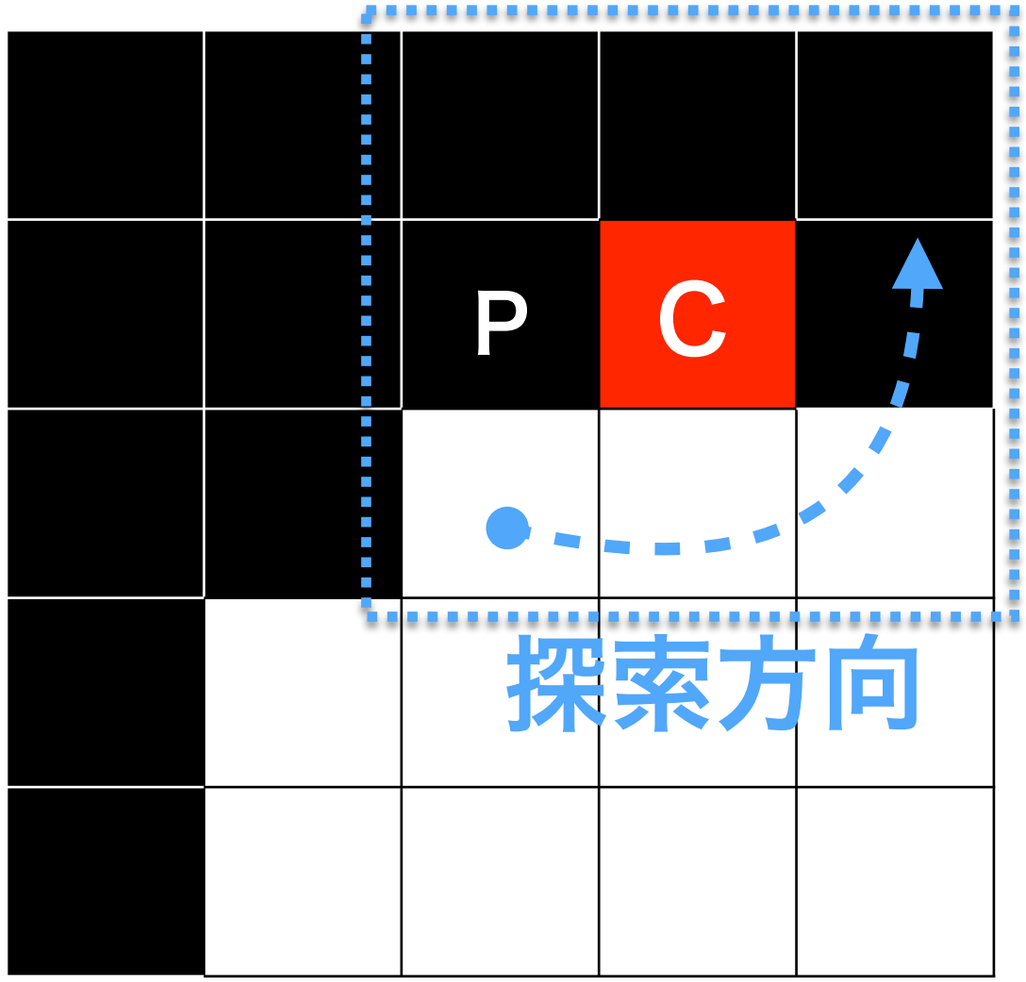
✓または2.で未探索の白画素が見つからなかった場合

→終了

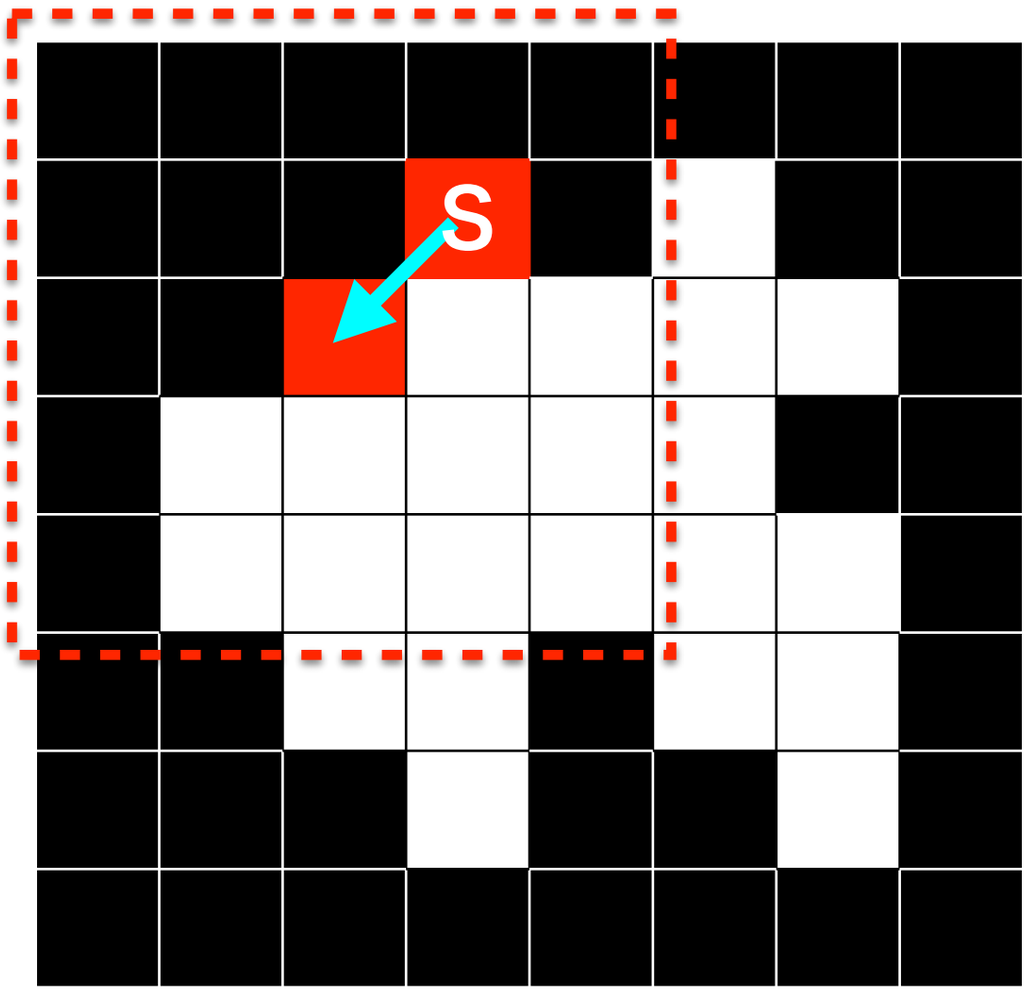
- 輪になっていない（開いている）輪郭

試してみましょう

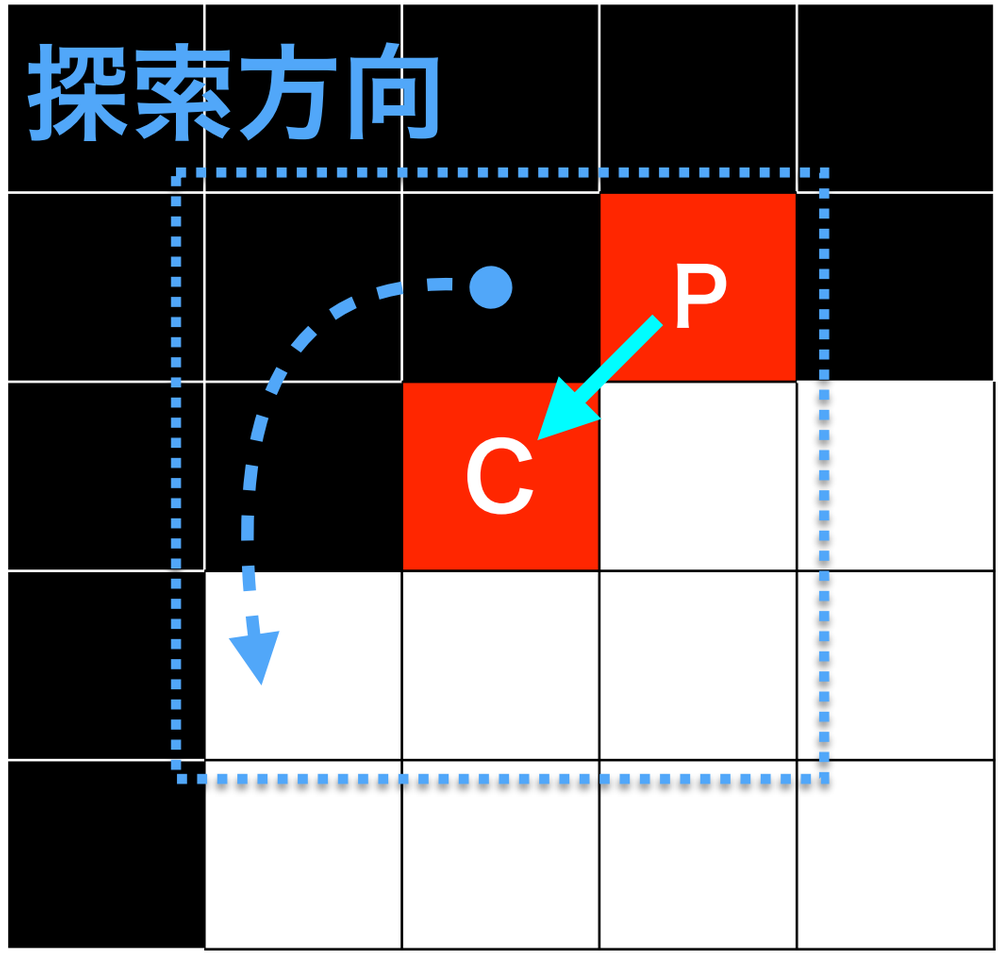




試してみましょう

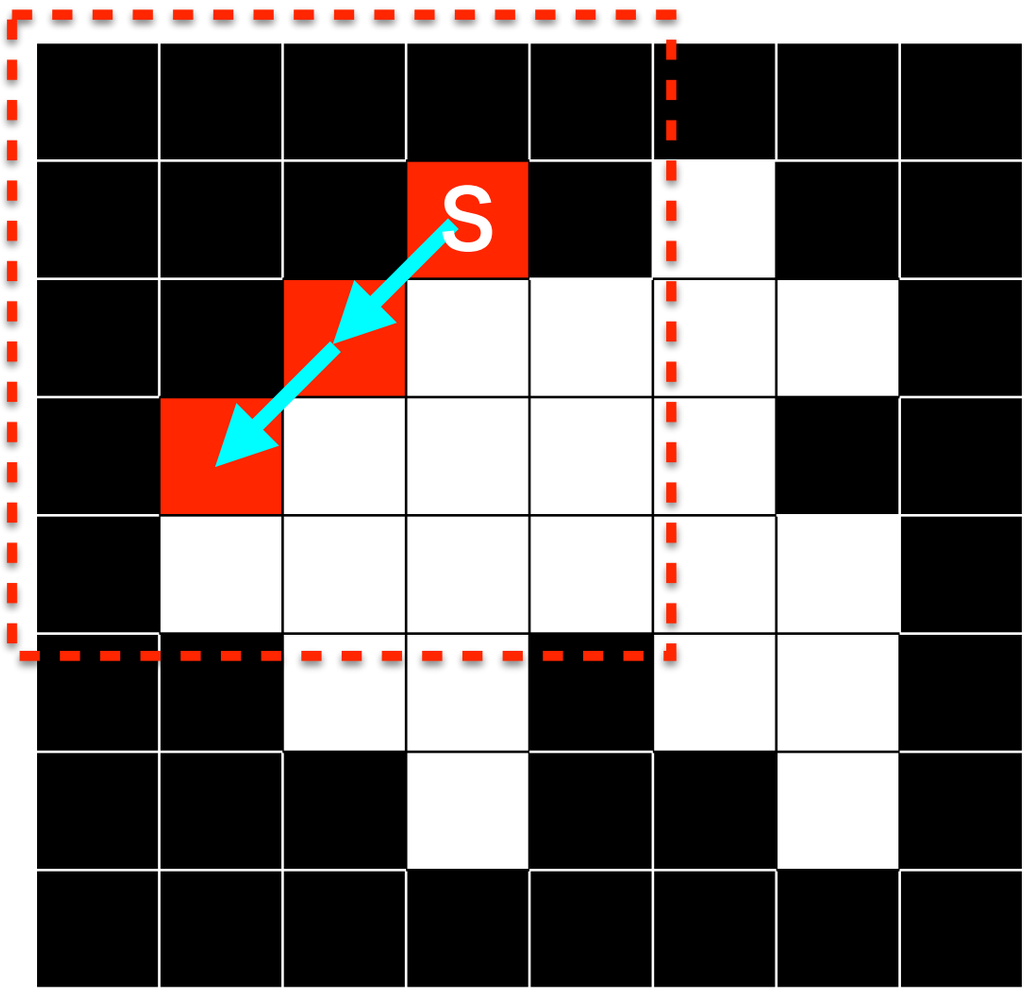


赤：追跡済

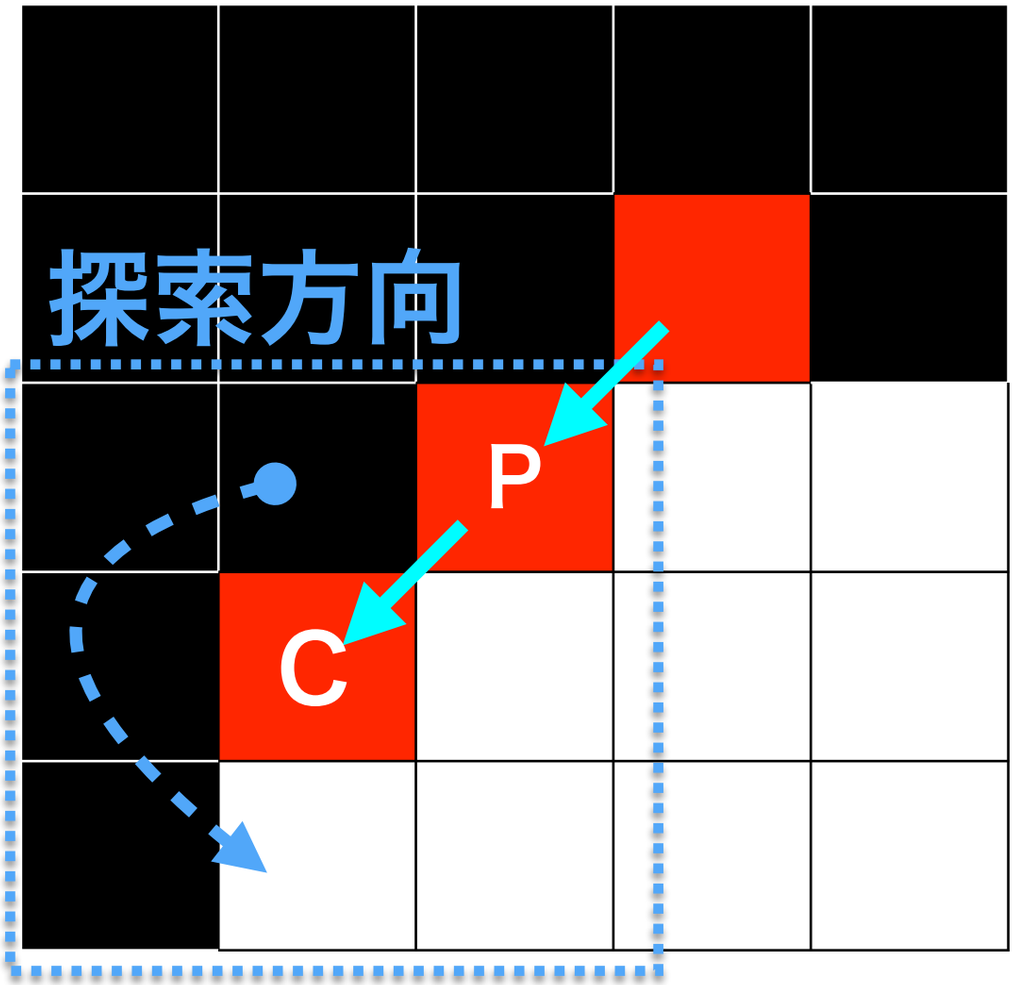


水色：輪郭

試してみましょう



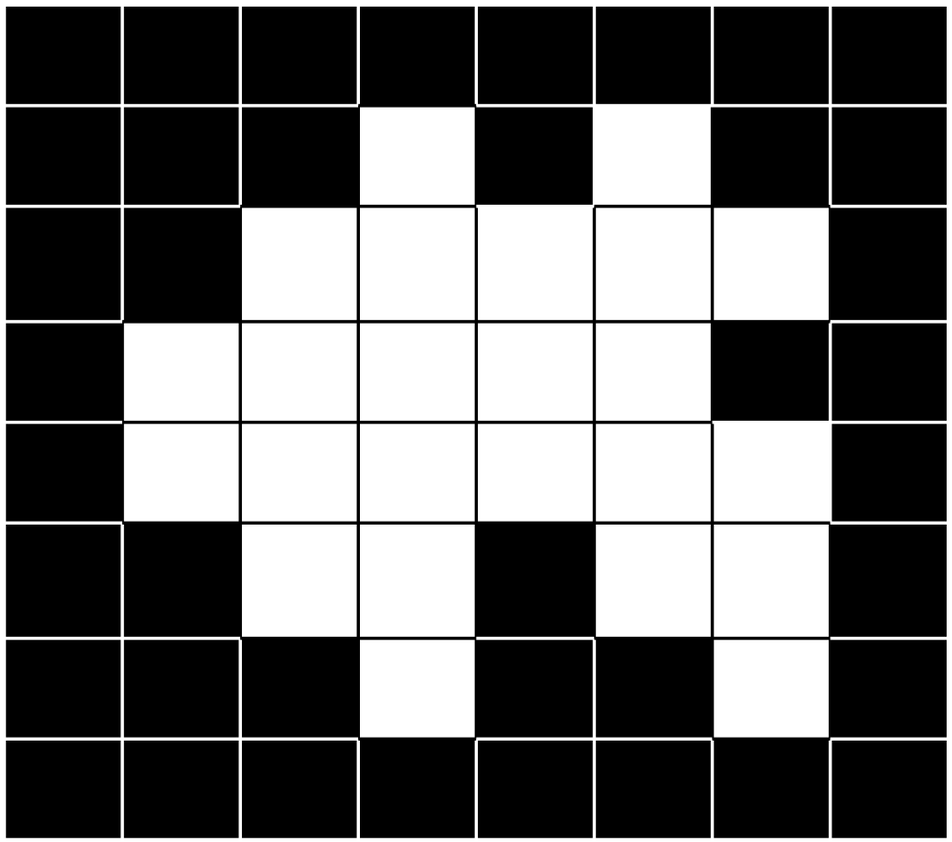
赤：追跡済



水色：輪郭

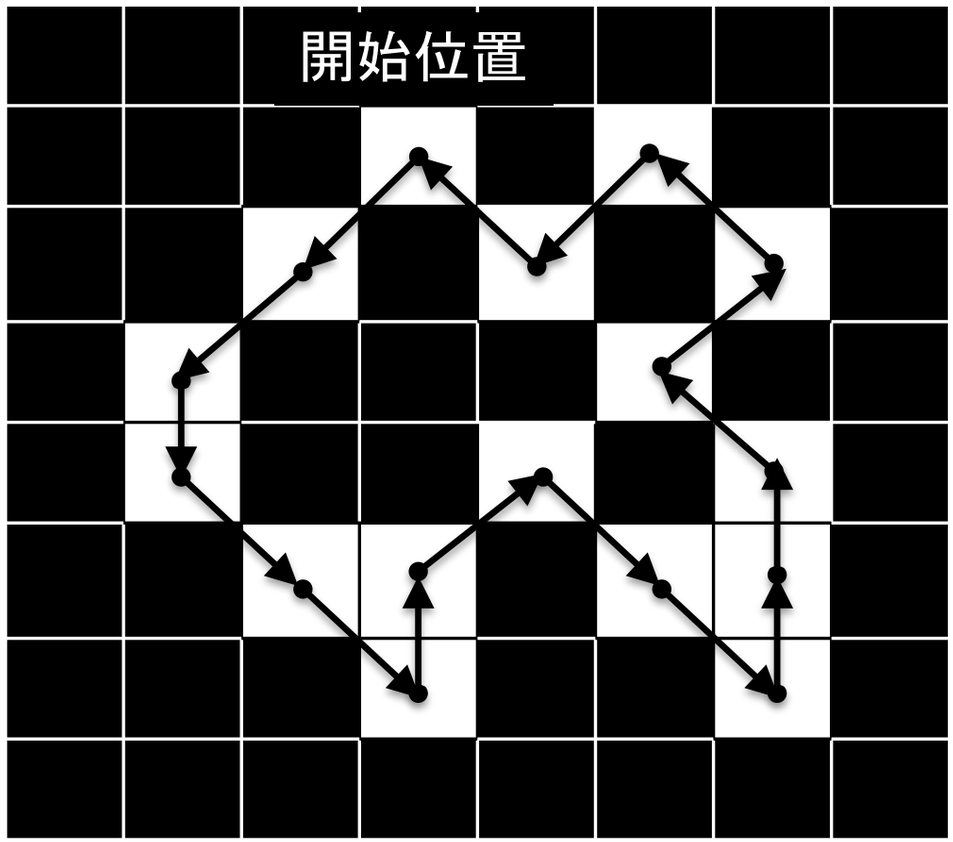
追跡結果

(教科書p.75)



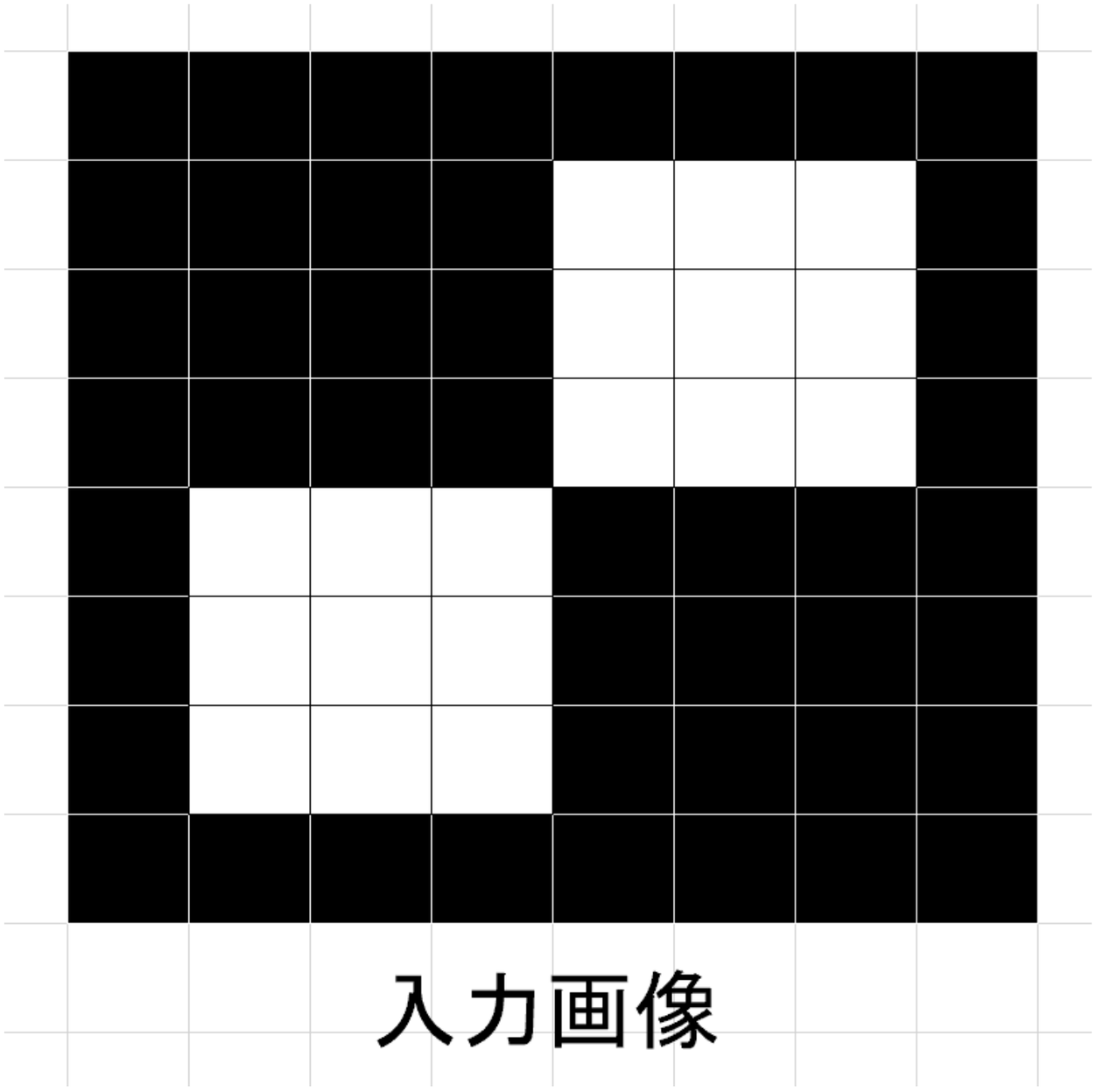
二値画像

輪郭
追跡



輪郭

輪郭追跡の演習

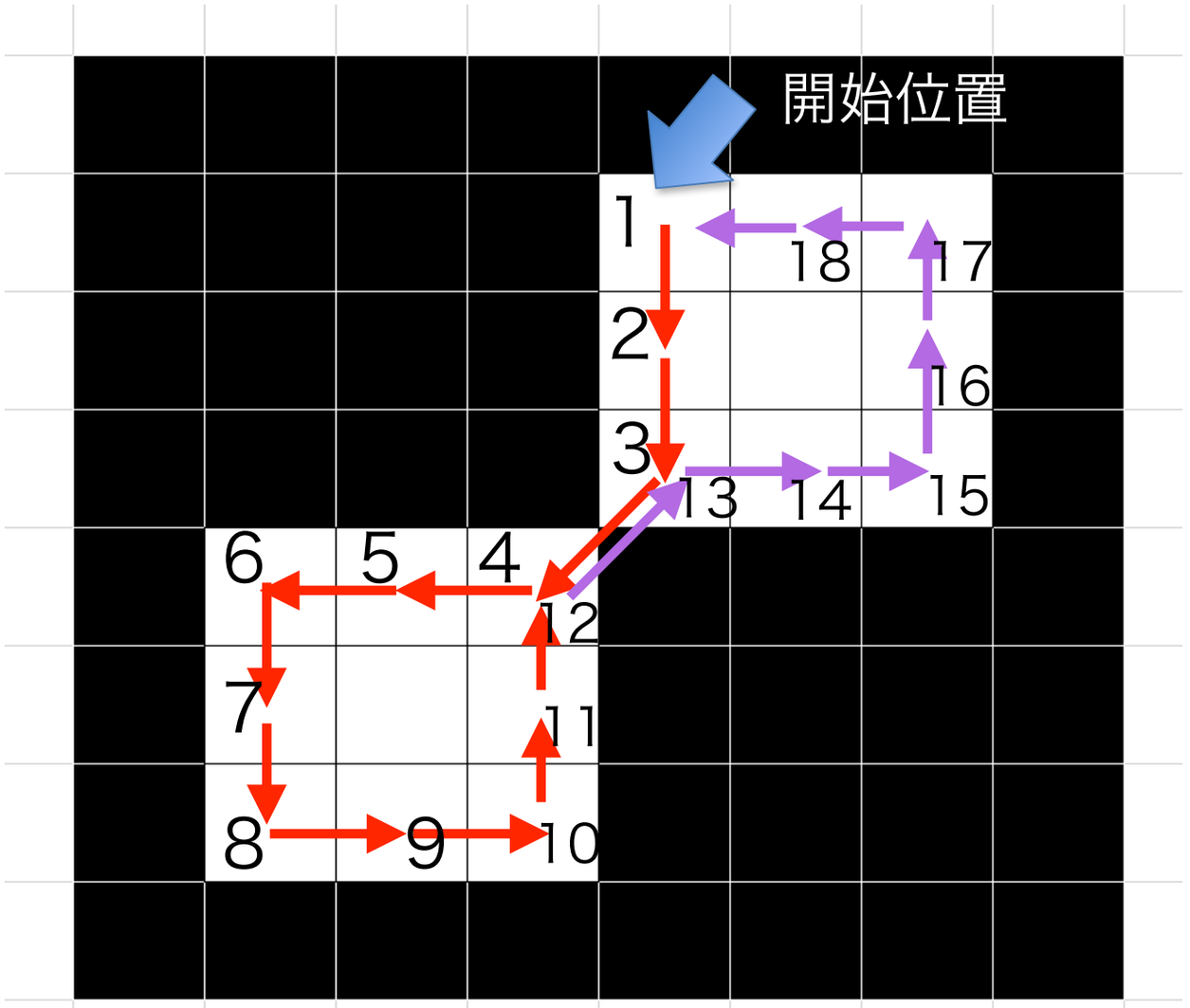


入力画像

Teamsのcontour_sample.xlsx/png をDL

追跡した画素の順に番号を記載してみましょう

輪郭追跡演習の解答



入力画像

輪郭の記述方法

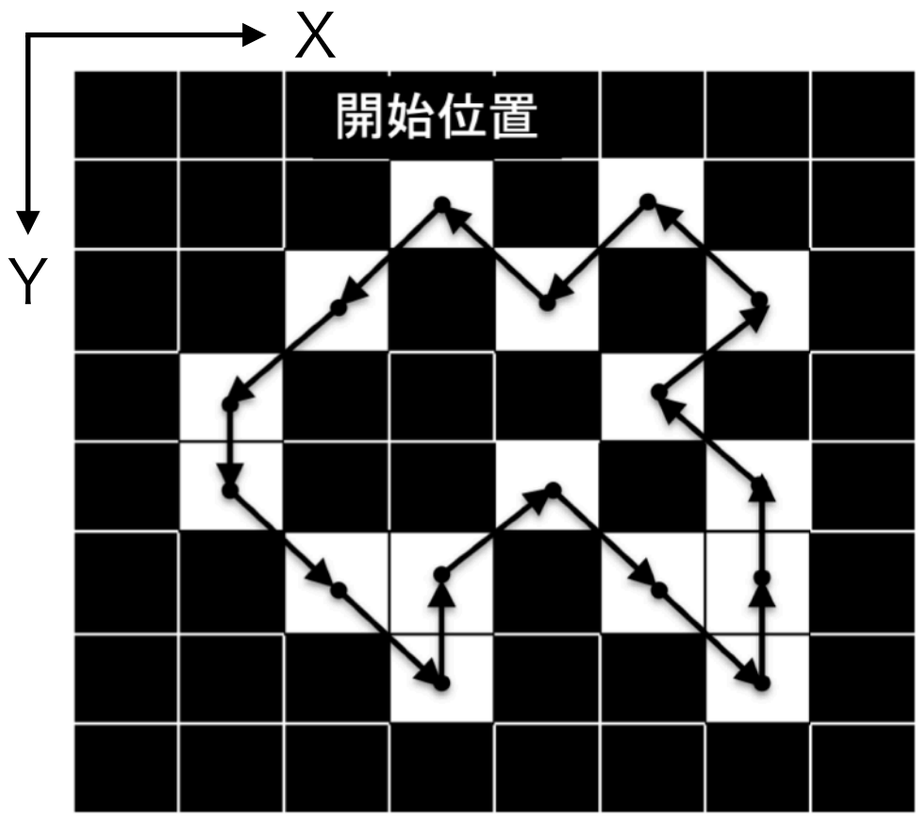
・求められた輪郭をどのように記述するか

- 輪郭画素の**座標位置**を追跡順に並べる

  ✓ わかりやすい・図形領域と対応づけしやすい

  ✓ 画像中の領域配置が変わると輪郭の座標位置も変わる

例



(3,1), (2,2), (1,3), (1,4),
(2,5), (3,6), (3,5), (4,4),
(5,5), (6,6), (6,5), (6,4),
(5,3), (6,2), (5,1), (4,2)

チェーンコード

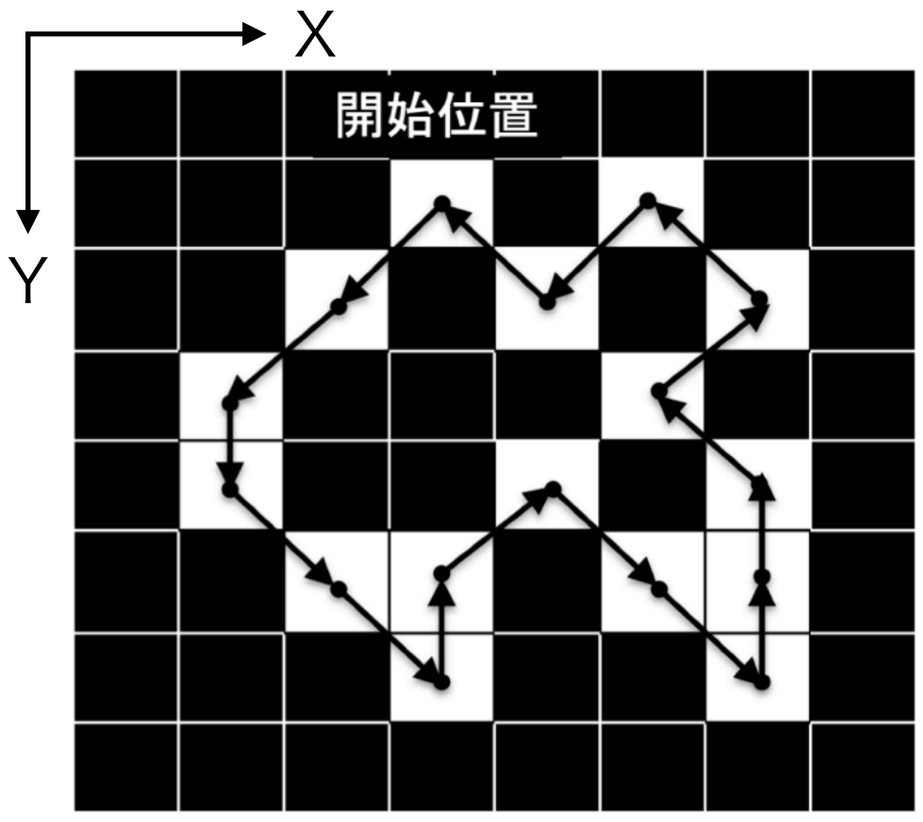
・求められた輪郭をどのように記述するか

- 輪郭画素の**追跡方向**を追跡順に並べる
  - ✓ 配置を変えても変わらない
  - ✓ データ数も少ない

3	2	1
4	C	0
5	6	7

C：注目画素
1～7：追跡方向

例



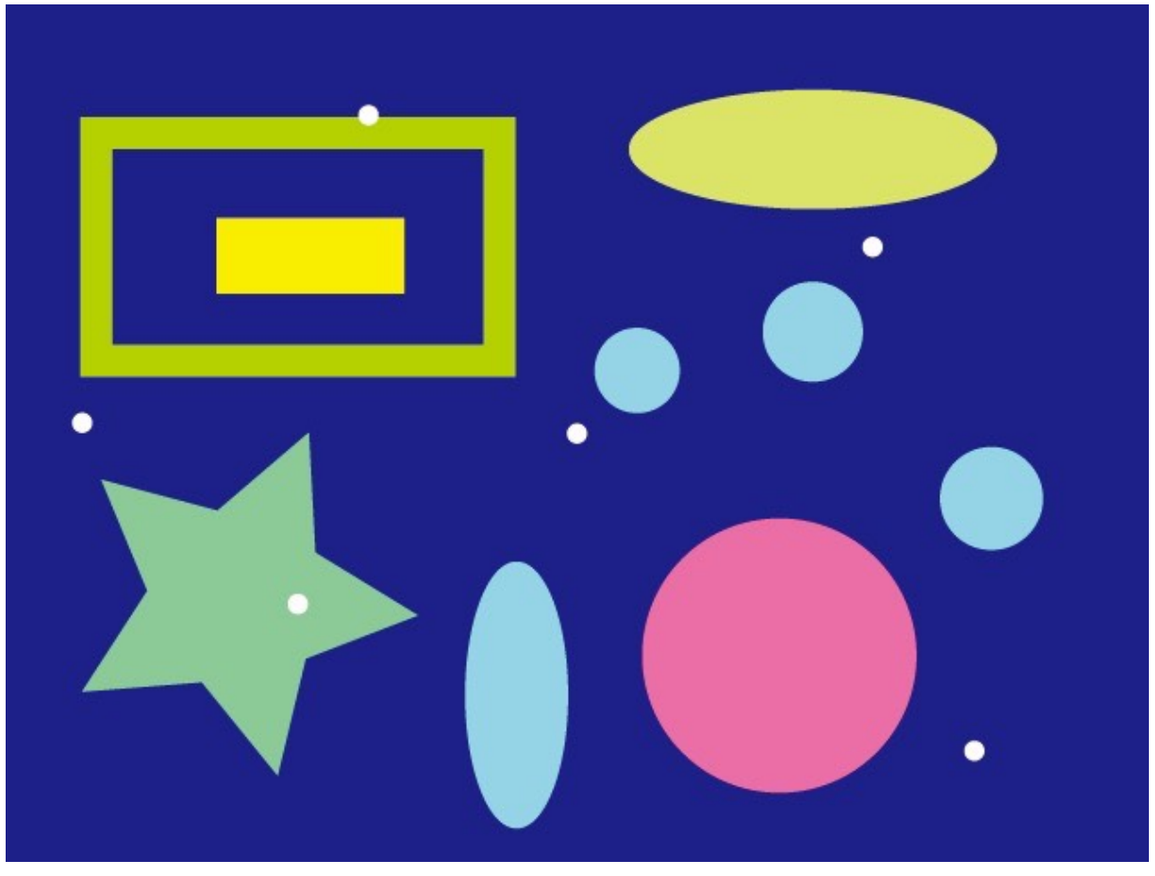
開始位置(3,1)
チェーンコード：
{5, 5, 6, 7, 7, 2, 1, 7,
7, 2, 2, 3, 1, 3, 5, 3}

輪郭追跡プログラムの演習

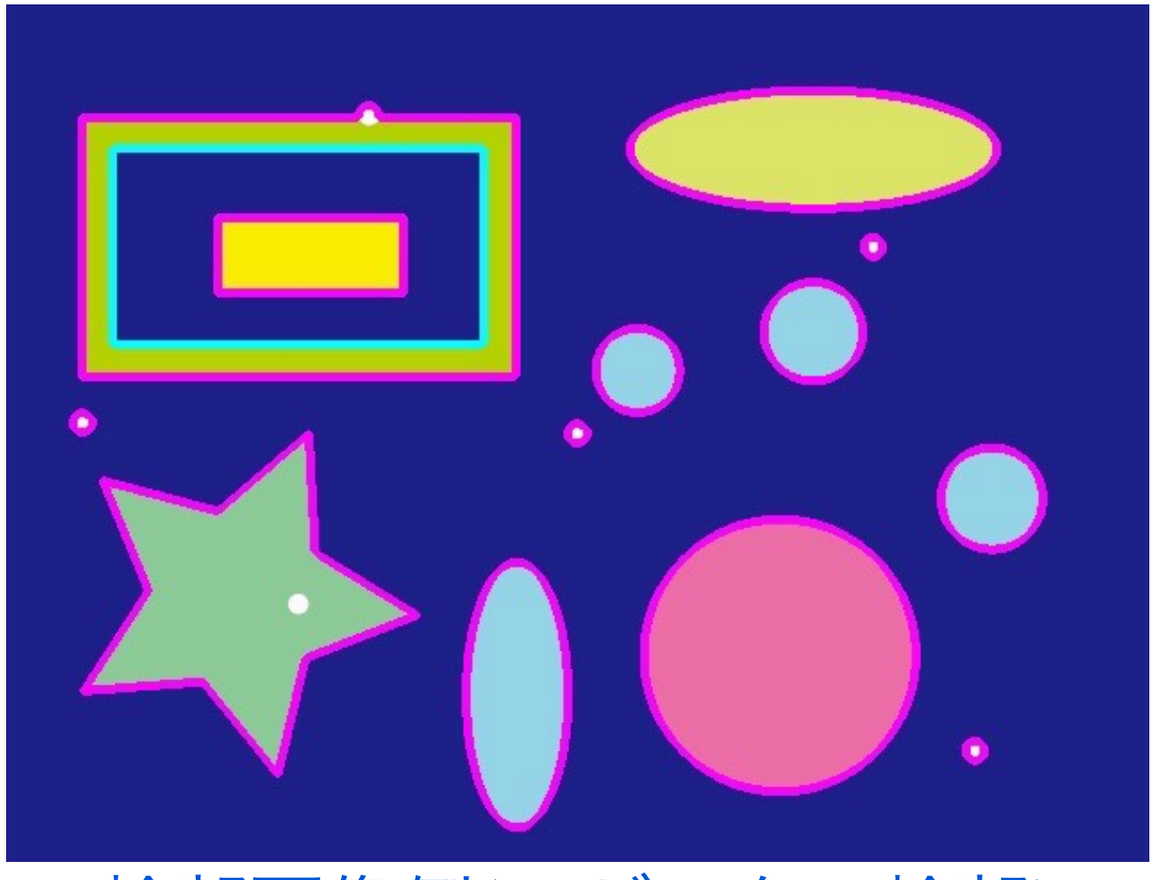
・プロジェクト名: contour4student

- Teamsから雛形ソースコードcontour4student.cpp と画像（sample.jpg）を入手

・カラー画像を入力し輪郭を抽出・描画する



入力画像



輪郭画像例(マゼンタの輪郭)

処理の流れ

1. 画像の宣言
   (入力画像, グレースケール画像, 二値画像,
   一時的な画像, 出力画像)
2. 輪郭の座標リストの宣言
3. 画像の入力 (**カラー**で入力)
4. 入力画像を出力画像にコピー
5. グレースケール化
6. 二値化 (固定閾値で実装. しきい値: 100)
7. 輪郭追跡
8. 輪郭の描画
9. 表示

処理の流れ

1. 画像の宣言
   (入力画像, グレースケール画像, 二値画像, 一時的な画像, 出力画像)

5つの画像変数の役割

2. 輪郭の座標リストの宣言
3. 画像の入力 (**カラー**で入力)
4. 入力画像を出力画像にコピー
5. グレースケール化
6. 二値化 (固定閾値で実装. しきい値: 100)
7. 輪郭追跡
8. 輪郭の描画
9. 表示

処理の流れ

1. 画像の宣言
   (入力画像, グレースケール画像, 二値画像,
   一時的な画像, 出力画像)
2. 輪郭の座標リストの宣言 New!
3. 画像の入力 (**カラー**で入力)
4. 入力画像を出力画像にコピー New!
5. グレースケール化
6. 二値化 (固定閾値で実装. しきい値: 100)
7. 輪郭追跡
8. 輪郭の描画 New!
9. 表示

まずは6. 二値化まで実装します

二値化まで

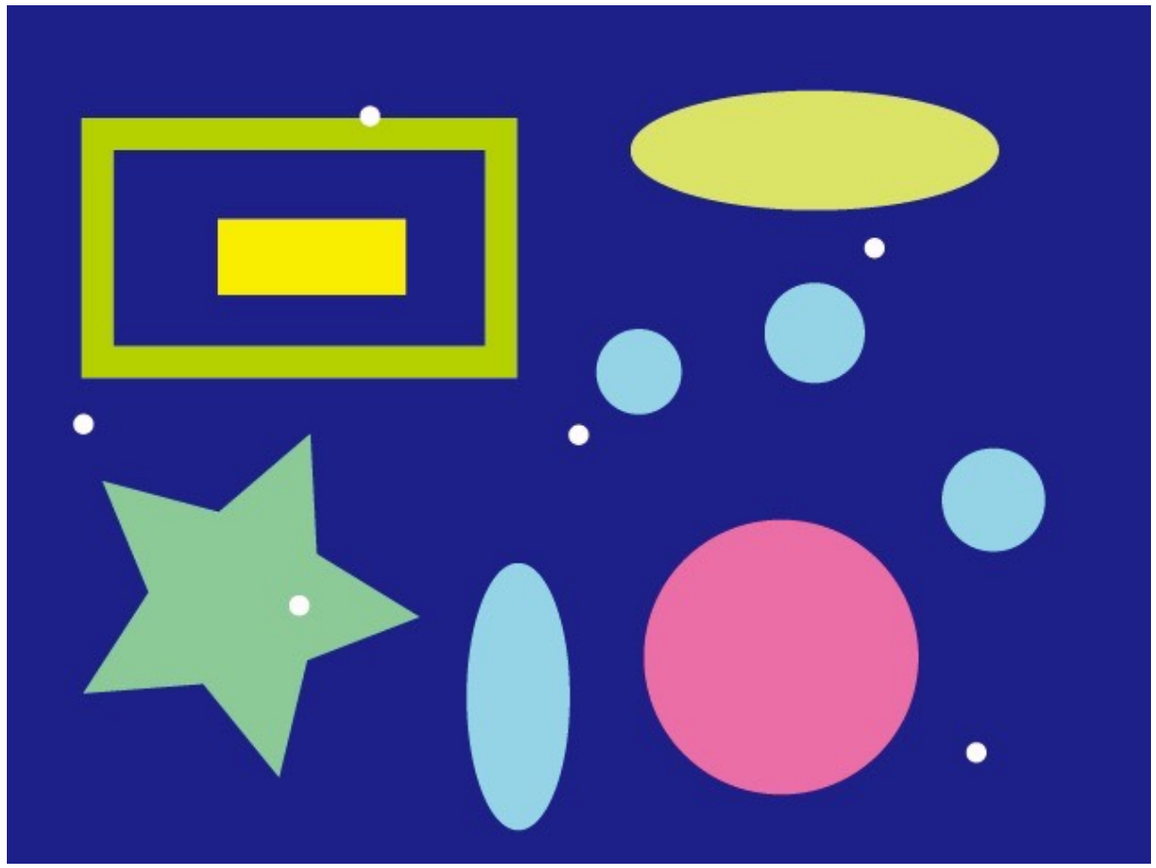
```
//4. 入力画像を出力画像にコピー（New!）
dst_img = src_img.clone();

//5. グレースケール化
cv::cvtColor(src_img, gray_img, cv::COLOR_BGR2GRAY);

//6. 二値化（固定しきい値で実装. しきい値: 100）
cv::threshold(gray_img, bin_img, BIN_TH, 255, cv::THRESH_BINARY);
//二値画像コピー（New!）
tmp_img = bin_img.clone();
```

・BIN_TH：二値化のしきい値 (100)　（#defineで定義）

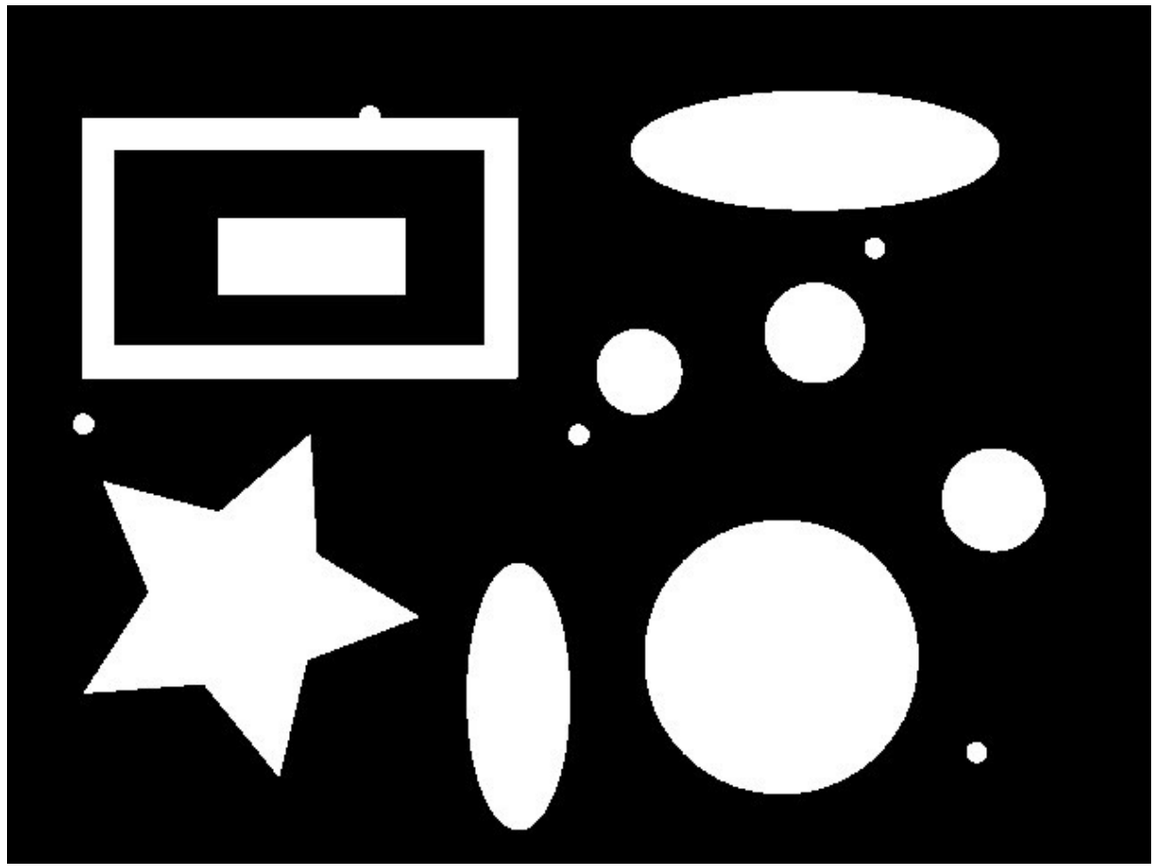
・**.clone()** ：画像のコピーを作成するメンバ関数

輪郭追跡で画像が加工されてしまうため,
二値画像bin_imgをtmp_imgにコピーして用いる

二値画像



入力画像



二値画像

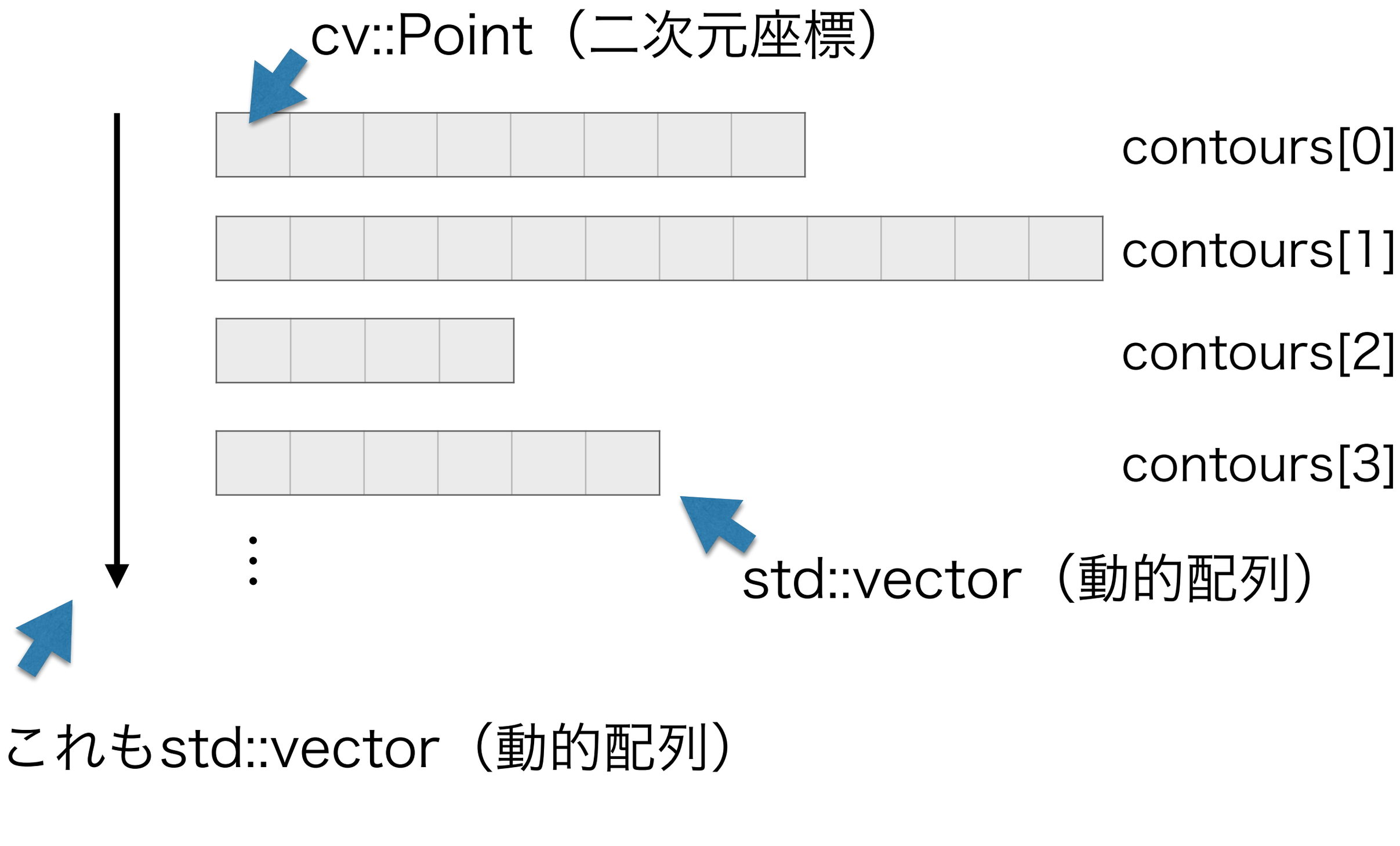
メモ

輪郭の座標リストの宣言

```
std::vector< std::vector< cv::Point > > contours;
```

- std::vector: 動的配列
  - 配列の大きさが固定されない（可変長）
- cv::Point: 座標
  - 二次元の座標
- **座標位置の 並び（＝輪郭）**をリスト化
  (cv::Point) の (std::vector) の (std::vector)

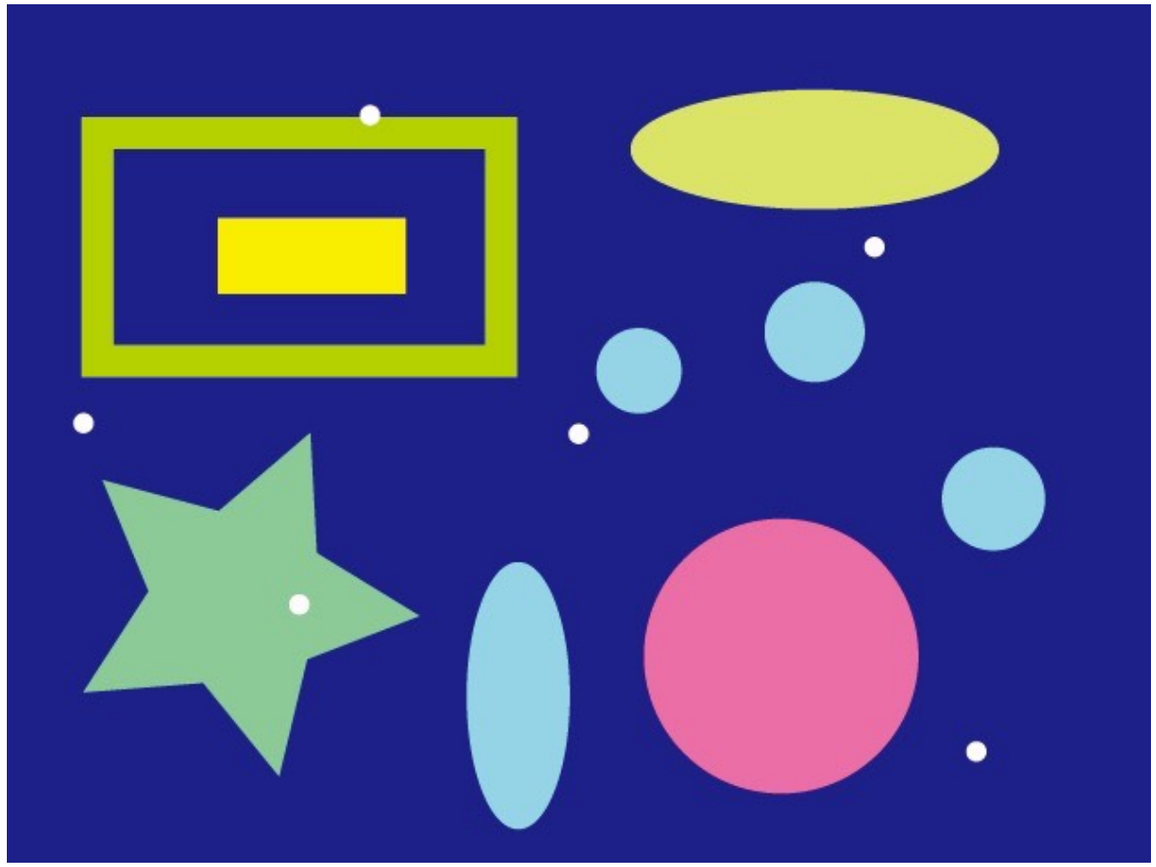
contoursの構造



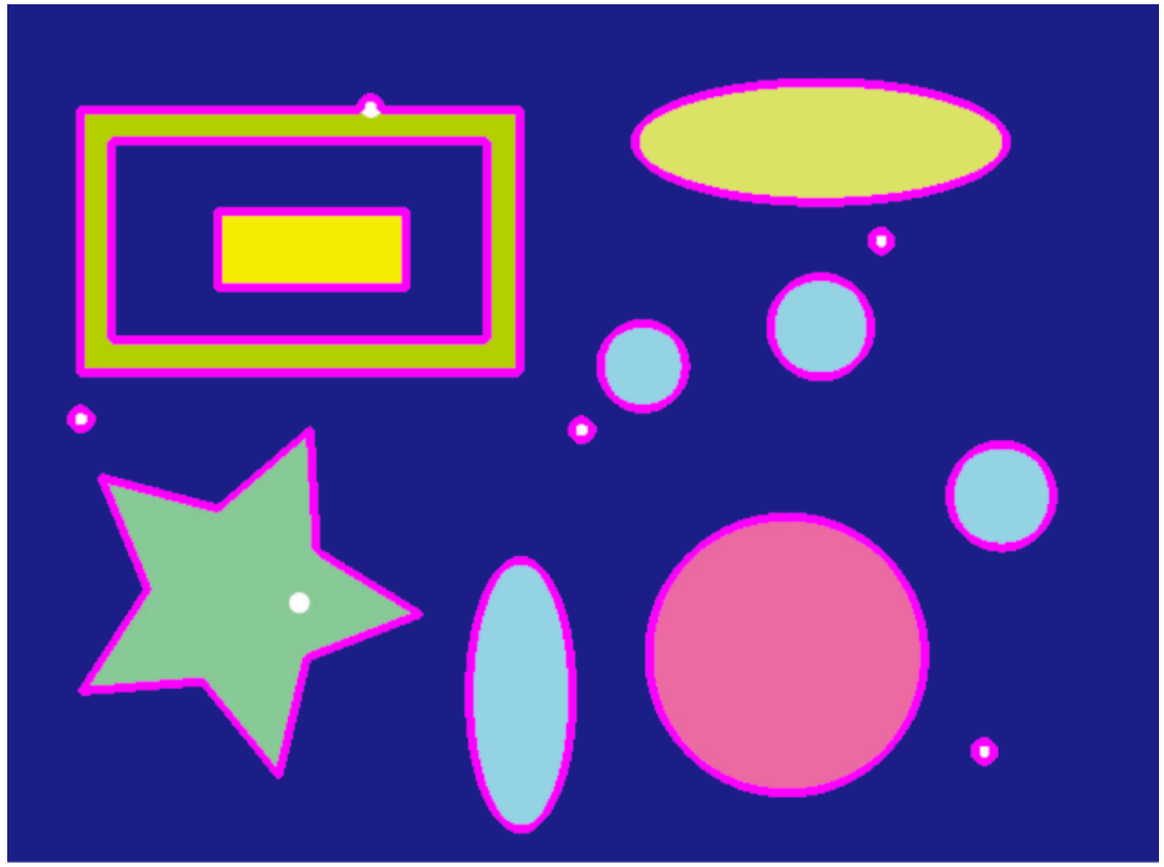
輪郭追跡から描画まで

```
//7. 輪郭追跡 (New!)
cv::findContours(tmp_img, contours,
           cv::RETR_LIST, cv::CHAIN_APPROX_NONE);

//8. 輪郭の描画 (New!)
for (int i=0; i<contours.size(); i++) {
    //輪郭の描画
    cv::drawContours(dst_img, contours, i, CV_RGB(255, 0, 255), 3);
}
```

描画された輪郭



入力画像



輪郭画像
（外輪郭・内輪郭全て）

メモ

輪郭追跡関数

- findContours：入力二値画像の輪郭を追跡する

```
cv::findContours(二値画像, 輪郭, 追跡モード, 輪郭近似手法);
```

- 今回の記載例

```
//7. 輪郭追跡(New!)
cv::findContours(tmp_img, contours,
   cv::RETR_LIST, cv::CHAIN_APPROX_NONE);
```

- cv::RETR_LIST: すべての輪郭追跡、リスト出力
- cv::CHAIN_APPROX_NONE: 8近傍、近似なし

輪郭追跡結果がcontoursに格納される

メモ

各輪郭へのアクセス方法

- for文でcontoursの各要素（各輪郭）にアクセス

```
//8. 輪郭の描画(New!)
for (int i=0; i<contours.size(); i++) {


}
```

- contours.size()：動的配列contoursのサイズ
  （輪郭が何個あるか）

  ✳contours[i]　でｉ番目の輪郭にアクセスできる
  （このプログラムではこの記載は使いません）

メモ

輪郭描画関数

- drawContours：輪郭を描画する

```
cv::drawContours(出力画像, 輪郭情報, 輪郭番号,
                                    輪郭の色, 描画用の線幅);
```

※線幅を負の値にすると輪郭内部も塗りつぶす

- 今回の記載例

```
//輪郭の描画
cv::drawContours(dst_img, contours, i,
                                CV_RGB(255, 0, 255), 3);
```

※CV_RGBで色を指定　B=R=255でマゼンタとしている

- この描画文をfor文の中に入れる

➡ 輪郭番号を指定して（この場合 i）描画する

contour4student.cpp 追記箇所まとめ

```
//4. 入力画像を出力画像にコピー (New!)
dst_img = src_img.clone();

//5. グレースケール化
cv::cvtColor(src_img, gray_img, cv::COLOR_BGR2GRAY);

//6. 二値化 (固定しきい値で実装. しきい値: 100)
cv::threshold(gray_img, bin_img, BIN_TH, 255, cv::THRESH_BINARY);
//二値画像コピー (New!)
tmp_img = bin_img.clone();

//7. 輪郭追跡 (New!)
cv::findContours(tmp_img, contours,
          cv::RETR_LIST, cv::CHAIN_APPROX_NONE);

//8. 輪郭の描画 (New!)
for (int i=0; i<contours.size(); i++) {
    //輪郭の描画
    cv::drawContours(dst_img, contours, i, CV_RGB(255, 0, 255), 3);
}
```

ノート

領域特徴量

(教科書p.118)

・輪郭追跡で閉輪郭が得られる

➡図形領域が定まる（領域抽出）

➡それぞれの図形領域に対する特徴を求めることができる

＝**領域特徴量**

・領域特徴量の種類

- 周囲長：輪郭の長さ
- 面積：領域の面積（輪郭で囲まれた面積）
- 円形度：どれだけ円に近いか
- 外接長方形（バウンディングボックス）：領域に接する最小長方形
- 重心　他

メモ

周囲長と面積

・輪郭に対する周囲長と面積を求める関数

```
cv::arcLength(輪郭, 閉輪郭か否か); //出力が周囲長
cv::contourArea(輪郭);             //出力が面積
```

・記載例

```
double L,S;
//周囲長（輪郭の長さ）
L = cv::arcLength(contours[i], true);
//面積
S = cv::contourArea(contours[i]);
```

i番目の輪郭を指定

メモ

円形度

・図形領域がどれだけ円に近いか

・領域の面積をＳ，周囲長をＬとすると
　円形度Ｒは以下の式で求められる：

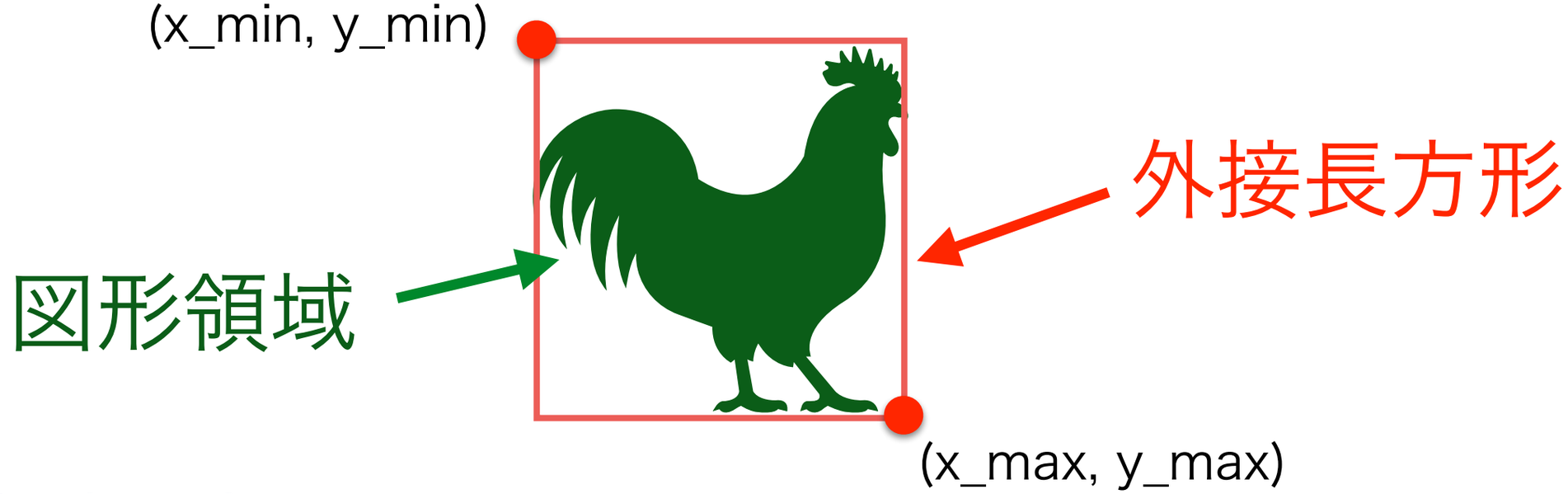
$$R = \frac{4\pi S}{L^2}$$

・理想的な円のとき，R=1

メモ

外接長方形 (バウンディングボックス)

・図形領域に接する最小の長方形



・求め方（原理）

- 図形領域の輪郭を求める
- 輪郭各画素の座標位置(x,y)からxとyの最小値・最大値を求める(x_min, x_max, y_min, y_maxとする)

➡ 外接長方形の左上の座標は(x_min, y_min)
右下の座標は(x_max, y_max)

メモ

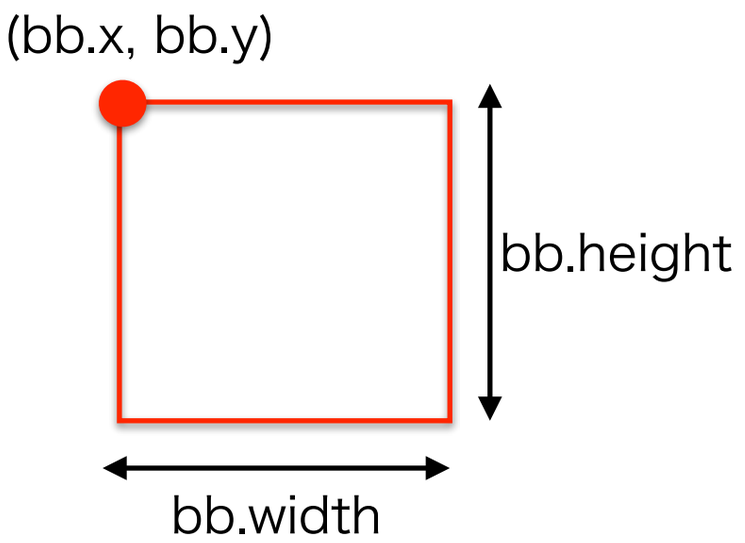
外接長方形用の関数

・外接長方形を求める関数

```
cv::boundingRect(輪郭); //出力が外接長方形
```

- 出力：構造体 Rect で定義された変数

➡ Rectのメンバ変数.x, .yは左上の座標,
.width, .heightは幅と高さを表す



・記載例

```
cv::Rect bb;
//外接長方形
bb = cv::boundingRect(contours[i]);
```

メモ

長方形描画関数

- rectangle：長方形を描画する

```
cv::rectangle(出力画像, 長方形, 色, 線幅);
```

- 長方形の指定方法　※線幅を負の値にすると内部も塗りつぶす

✓ 構造体Rectの変数（前ページのbb）

✓ 左上の頂点と右下の頂点(cv::Point型)を並べて記載

- 記載例

```
//外接長方形の描画
cv::rectangle(dst_img, bb,
                        CV_RGB(255, 0, 255), 3);
```

※CV_RGBで色を指定　B=R=255でマゼンタとしている

領域特徴量の演習

・プロジェクト名: contour2

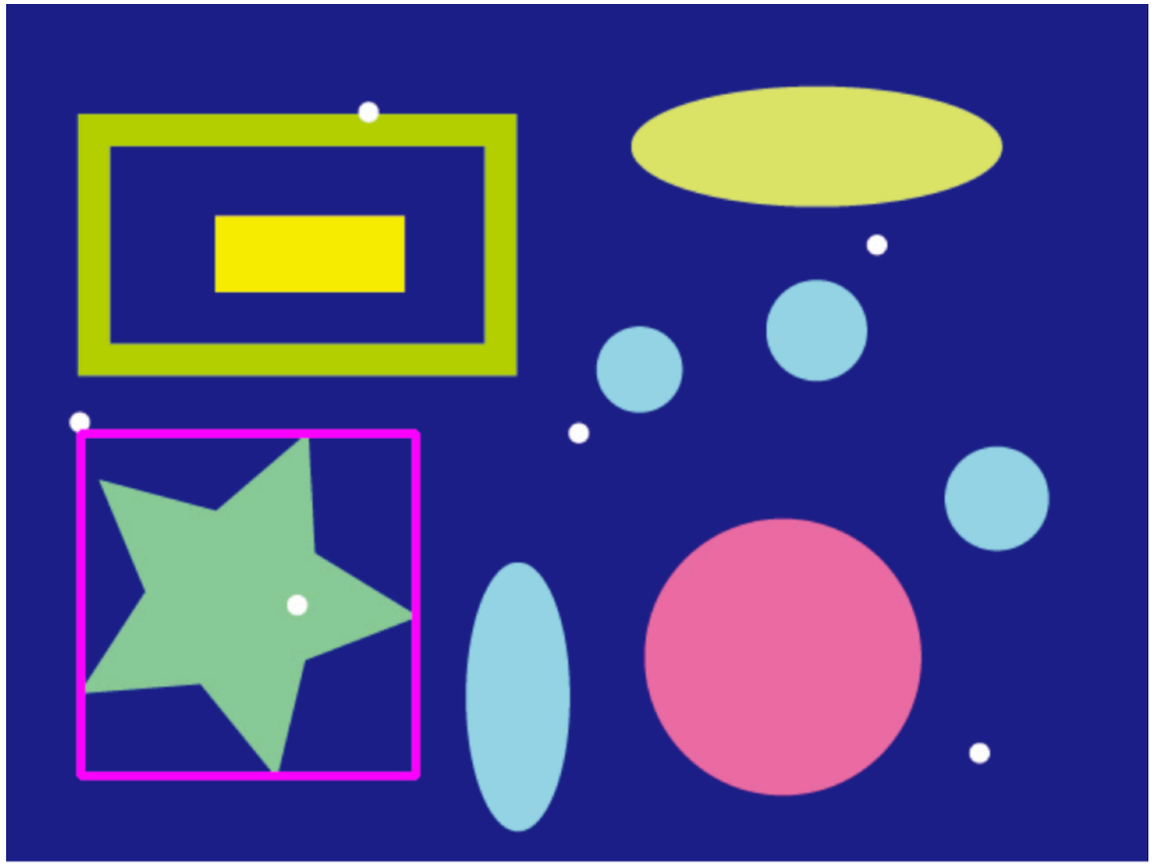
- 先ほどのcontour4student.cpp（追記済）をコピーして新しいプロジェクトを作成　入力画像も同じ

・指定輪郭の特徴量を取得して表示・描画する

```
輪郭数=14
輪郭番号？(0～13):4
周囲長:702.482317, 面積:14900.500000
```

番号を入力

輪郭番号を指定すると
周囲長と面積を表示する
（上は4を入力した例）



指定輪郭の外接長方形を描画(マゼンタ)

処理の流れ

1. 画像の宣言
   (入力画像, グレースケール画像, 二値画像,
   一時的な画像, 出力画像)
2. 輪郭の座標リストの宣言
3. 画像の入力 (**カラー**で入力)
4. 入力画像を出力画像にコピー
5. グレースケール化
6. 二値化 (固定閾値で実装. しきい値: 100)
7. 輪郭追跡
8. ~~輪郭の描画~~ 領域特徴量の取得・表示・描画
   前のコードをコメントアウトして新たにコードを作成
9. 表示

まずはコメントから

```
//8.1 輪郭数の表示

//8.2 表示する輪郭番号の入力

//8.3 領域特徴量の取得・表示

//8.4 外接長方形の取得

//8.5 外接長方形の描画
```

・この流れでコードを作成しましょう

- 入力・表示例：右
- 外接長方形の描画は輪郭と同じくマゼンタ、太さ３で

```
輪郭数=14
輪郭番号？(0～13):4
周囲長:702.482317, 面積:14900.500000
```

入力・表示例

領域特徴量取得・表示・描画

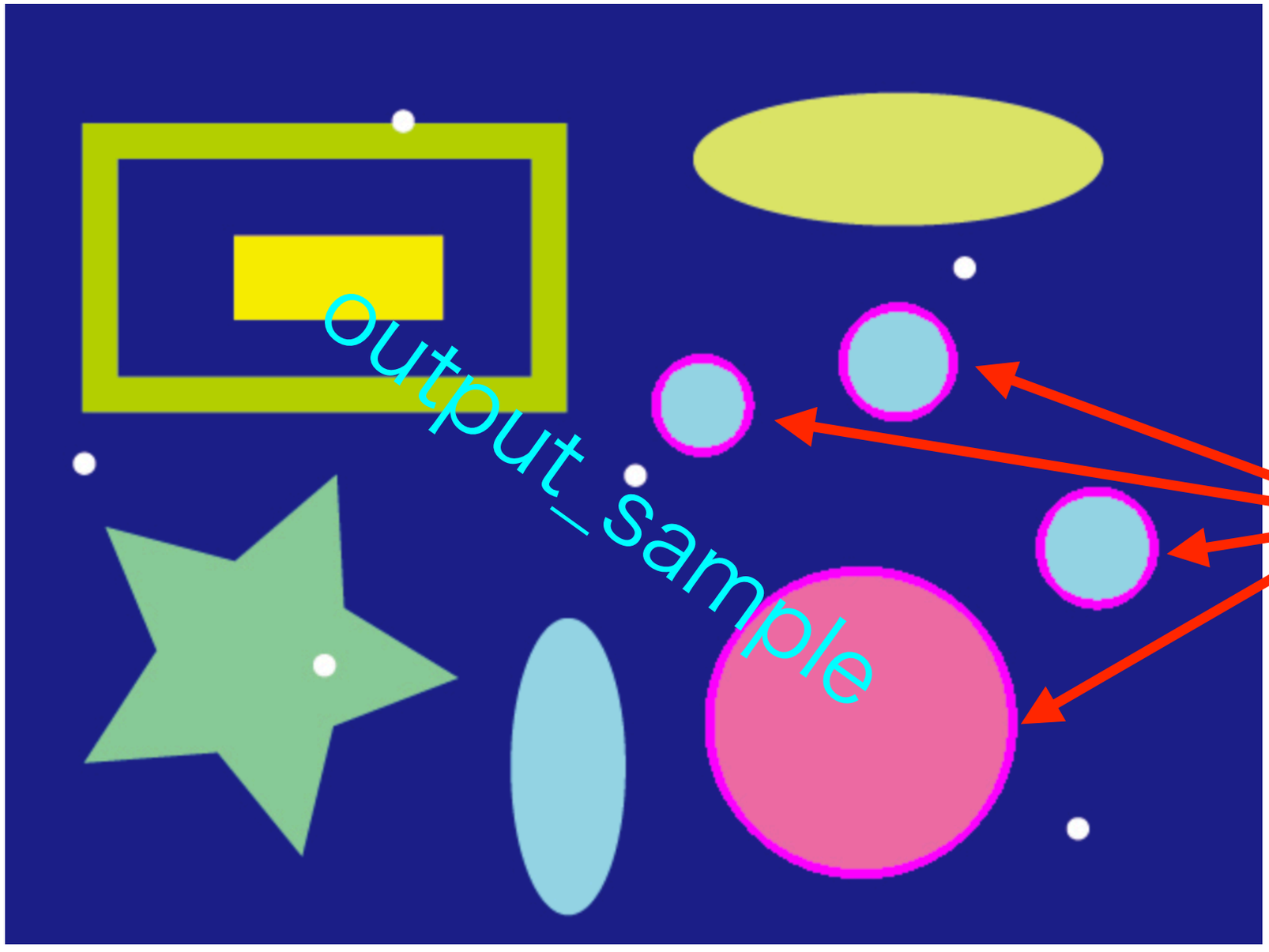
```
//8.1 輪郭数の表示
printf("輪郭数=%d\n", (int)contours.size());
//8.2 表示する輪郭番号の入力
int i;
printf("輪郭番号？(0～%d):", (int)contours.size()-1);
scanf("%d",&i);
//8.3 領域特徴量の取得・表示
double L,S;
L = cv::arcLength(contours[i], true);
S = cv::contourArea(contours[i]);
printf("周囲長:%lf, 面積:%lf\n",L,S);
//8.4 外接長方形の取得
cv::Rect bb = cv::boundingRect(contours[i]);
//8.5 外接長方形の描画
cv::rectangle(dst_img, bb, CV_RGB(255,0,255), 3);
```

課題1

- 演習で用いた入力画像sample.jpgに対して、
  円の輪郭だけを描画するプログラムを作成し、
  結果画像を出力せよ
  （ただし小さな白色の円は除く）

- 提出ファイル
  - ソースプログラム：09_01_contour_学籍番号.cpp
  - 出力画像：09_01_output_学籍番号.jpg（カラー画像
    jpgでなくてもよい）

課題１の出力画像例



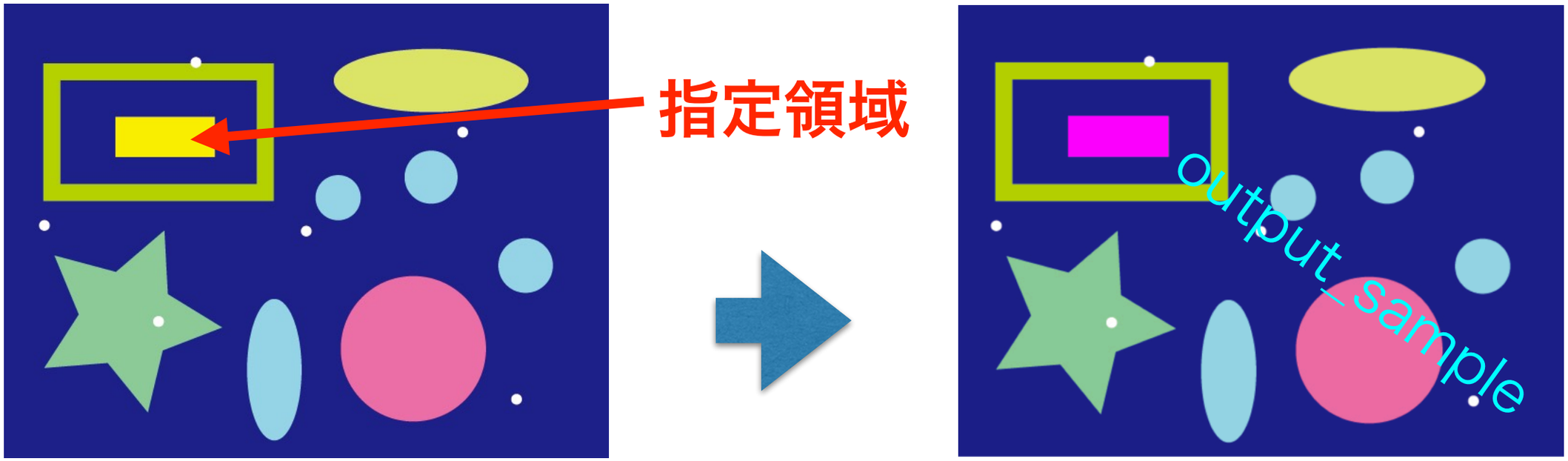
課題1のヒント

- **まず輪郭を求める**
- 各輪郭に対し**面積**を求める
- 周囲長も求めて、各輪郭の**円形度**を算出する
  - 円周率πは 定数 M_PI で用いることができる
- 面積と円形度が**条件を満たすときだけ**輪郭を描画する
  - ある程度面積が大きな輪郭を対象にすれば白色の円を除去できる
  - 円かどうかの判定は 円形度 ≧ 0.8 程度でよい
- 出力画像には予め入力画像をコピーしておく
  - 輪郭を上書きする

課題２（提出自由・加点対象）

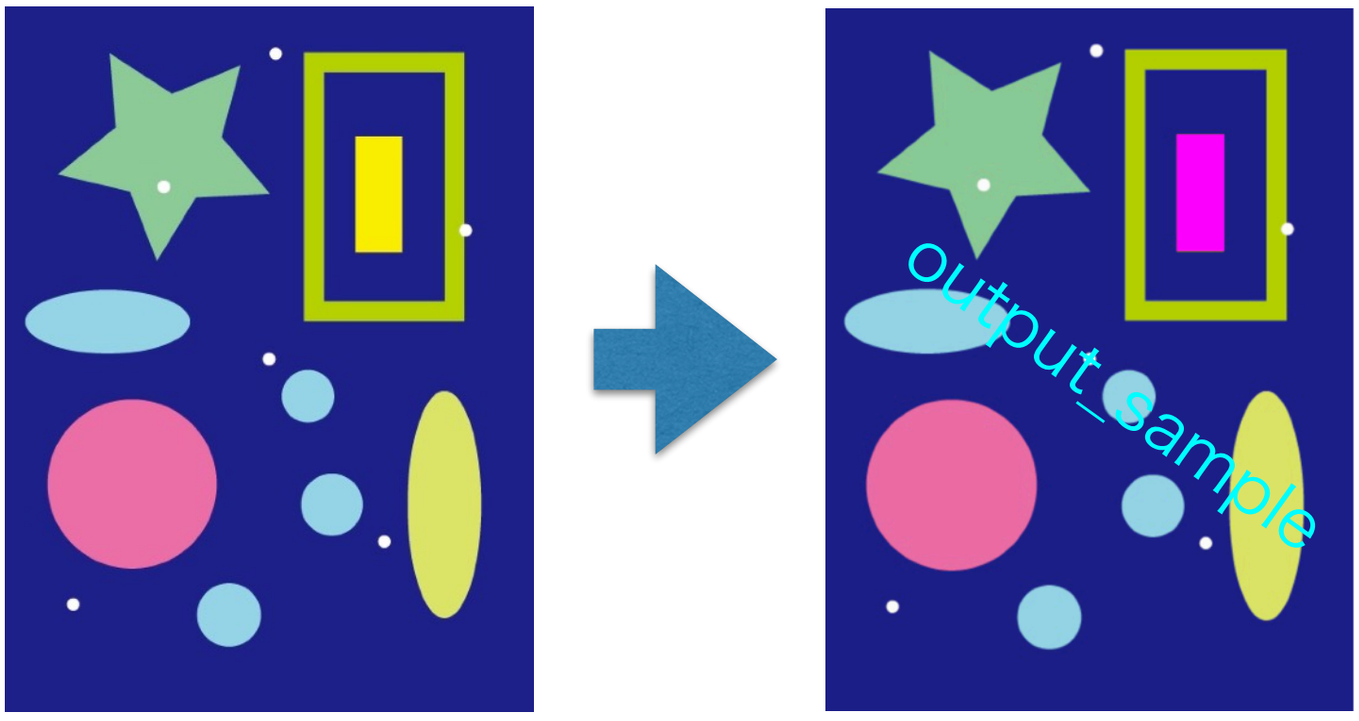
- 課題１と同じ入力画像sample.jpgに対して、
  領域特徴量を用いて**指定された図形領域を**
  マゼンタで塗り潰すプログラムを作成し、
  結果画像を出力せよ　※(R,G,B)=(255,0,255)でマゼンタ
  - 但し、90度回転したsample_rotate.jpgを入力画像としても、**同じプログラムで**同じ領域が塗りつぶされること
- 提出ファイル
  - ソースプログラム：09_02_rectangle_学籍番号.cpp
  - 出力画像：09_02_output_学籍番号.jpg（カラー画像jpgでなくてもよい）

課題２の指定領域と出力画像例



sample.jpg

出力画像例



sample_rotate.jpg

sample_rotate.jpgでも
同じプログラムで
同じ領域が塗りつぶされる

課題2のヒント

- 課題１と同じように輪郭追跡から領域特徴量を求める
- 塗り潰しはdrawContoursで線幅を-1（負の値）にすれば実施される
- 指定領域だけを描画するための領域特徴量と条件を，
  試行錯誤して決める
  - 例
    - ✓ 円形度と面積を用いる
    - ✓ 外接長方形の幅・高さを用いると良い
    - ✓ 複数の条件分岐で判定する（どれか一つだけだと難しい）
    - ✓ 円形度は長方形だけを選別するためには当てにならない
      （円の除去には使えるが楕円は残ることが多い）

感想と要望

- 感想や要望をお知らせください