```
/*****************/
/* main_process  */
/*****************/

// gRed, gGreen, gBlueの2次元配列にクエリ画像のRGB情報が格納されています
// 同じサイズのgBuff_red, gBuff_green, gBuff_blueに結果画像情報を格納してリターンします

#define AREA_INFO_MAX 1000  // area_info_counterの最大値
#define AREA_INFO_CONTENTS 13  // area_info[][]の項目数

// エリアトレース時の有効リアの条件を指定します
// ここではコインの直径である縦横双方とも200～300ピクセルのエリアのみ有効にするように設
定
#define COIN_WH_MIN 200
#define COIN_WH_MAX 300
int condition_func_coin(int top, int bottom, int left, int right, int trace_counter,
int pxl_counter) {
  if (COIN_WH_MIN <= bottom - top + 1 && bottom - top + 1 <= COIN_WH_MAX &&
    COIN_WH_MIN <= right - left + 1 && right - left + 1 <= COIN_WH_MAX)  return 1;
  else return 0;
}

// エッジ部分を検出してその外形をトレースします
int main_process(unsigned char** gRed, unsigned char** gGreen, unsigned char** gBlue,
  unsigned char** gBuff_red, unsigned char** gBuff_green, unsigned char** gBuff_blue,
  int width_pixel, int height_pixel) {

  // gGray[][]メモリ確保
  int i;
  unsigned char** gGray = (unsigned char**)malloc(sizeof(unsigned char*) *
height_pixel);
  if (gGray == NULL)  return -99;
  unsigned char* gGray_p = (unsigned char*)malloc(sizeof(unsigned char) * height_pixel
* width_pixel);
  if (gGray_p == NULL) {
    free(gGray);
    return -99;
  }
  for (i = 0; i < height_pixel; i++) {
    gGray[i] = gGray_p + (long long)i * (long long)width_pixel;
  }

  // グレースケールを生成します
  int j;
  for (i = 0; i < height_pixel; i++) {
    for (j = 0; j < width_pixel; j++) {
      gGray[i][j] = (gRed[i][j] + gGreen[i][j] + gBlue[i][j]) / 3;
    }
  }

  // エッジ部分をブルーにします
  int value = 255;
  int diff_distance = 2;
  int brt_diff_th = 15;
  int return_code = ABHB_edge_mapping(gGray, gBuff_blue, width_pixel, height_pixel,
    value, diff_distance, brt_diff_th);

  // area_info[AREA_INFO_MAX][AREA_INFO_CONTENTS]メモリを確保します
  int n;
  int** area_info;
  int* area_info_p;
  area_info = (int**)malloc(sizeof(int*) * AREA_INFO_MAX);
  if (area_info == NULL) {
```

```
    return -99;
  }
  area_info_p = (int*)malloc(sizeof(int) * AREA_INFO_MAX * AREA_INFO_CONTENTS);
  if (area_info_p == NULL) {
    free(area_info);
    return -99;
  }
  for (n = 0; n < AREA_INFO_MAX; n++) {
    area_info[n] = area_info_p + (long long)n * (long long)AREA_INFO_CONTENTS;
  }

  // 外周をトレースします
  int area_info_counter = 0;
  ABHB_area_trace(AREA_INFO_MAX, &area_info_counter, area_info, width_pixel,
height_pixel,
    gBuff_red, gBuff_green, gBuff_blue, condition_func_coin); //①

  // カレントディレクトリに画像をダンプします
  char dump_file_name[256];
  sprintf(dump_file_name, "trace.bmp");
  ABHB_bmp_read_write(1, dump_file_name, gBuff_red, gBuff_green, gBuff_blue,
&width_pixel, &height_pixel);

  // 他のエリアに完全に囲まれているエリアを無効にします
  int m;
  for (n = 0; n < area_info_counter; n++) {
    if (0 <= area_info[n][0]) {
      for (m = 0; m < area_info_counter; m++) {
        if (m != n && 0 <= area_info[m][0]) {
          if (area_info[m][4] <= area_info[n][4] && area_info[n][5] <= area_info[m][5]
&&
            area_info[m][6] <= area_info[n][6] && area_info[n][7] <= area_info[m][7]) {
            area_info[n][0] = -1;  // area_info[][0]=-1にすることによりこのエリアは無効
であるとします

            // 無効にしたエリアを囲む緑の矩形をクリアします
            for (i = area_info[n][4]; i <= area_info[n][5]; i++) {
              j = area_info[n][6];
              gBuff_green[i][j] = 0;
              j = area_info[n][7];
              gBuff_green[i][j] = 0;
            }
            for (j = area_info[n][6]; j <= area_info[n][7]; j++) {
              i = area_info[n][4];
              gBuff_green[i][j] = 0;
              i = area_info[n][5];
              gBuff_green[i][j] = 0;
            }
          }
        }
      }
    }
  }

  // area_info[][0]=-1を無効エリアとして、有効なエリアのみを残してソートします
  ABHB_sort_effective_area_info(&area_info_counter, area_info); //②

  // 有効エリアをリストします
  for (n = 0; n < area_info_counter; n++) {
    int area_w = area_info[n][7] - area_info[n][6] + 1;
    int area_h = area_info[n][5] - area_info[n][4] + 1;

    printf("  %2d(%4d,%4d)  area_w=%3d  area_h=%3d  pixel=%5d¥n",
```

```
        n, area_info[n][6], area_info[n][4], area_w, area_h, area_info[n][12]);
    }


    free(area_info_p);
    free(area_info);
    free(gGray_p);
    free(gGray);

    return 0;
}
```

****
area_info[n][k]

n：エリアNo.　0～area_info_counter - 1

area_info[n][0]　：　=0　有効/無効フラグ（通常無効にする場合 -1 にする）
area_info[n][1]　：　トレースピクセル数
area_info[n][2]　：　トレース開始ピクセルY座標
area_info[n][3]　：　トレース開始ピクセルX座標
area_info[n][4]　：　エリア上辺Y座標
area_info[n][5]　：　エリア下辺Y座標
area_info[n][6]　：　エリア左辺X座標
area_info[n][7]　：　エリア右辺X座標
area_info[n][8]　：　エリア上辺に接するX座標（複数点ある場合は中点）
area_info[n][9]　：　エリア下辺に接するX座標（複数点ある場合は中点）
area_info[n][10]　：　エリア左辺に接するY座標（複数点ある場合は中点）
area_info[n][11]　：　エリア右辺に接するY座標（複数点ある場合は中点）
area_info[n][12]　：　エリア内ブルーピクセル数

****関数***
ABHB_area_trace()

引数１：int area_info_counter_max　　エリア情報格納数最大値（メモリ確保した数）
引数２：int* area_info_counter　　エリア情報数ポインタ
引数３：int** area_info　　エリア情報2次元配列ポインタ
引数４：int width_pixel　　横方向ピクセル数
引数５：int height_pixel　　縦方向ピクセル数
引数６：unsigned char** gBuff_red　　トレースしたピクセルを255とします
引数７：unsigned char** gBuff_green　　有効エリアを囲む矩形の辺を255とします
引数８：unsigned char** gBuff_blue　　=255のピクセルの塊をトレースします
引数９：int(*condition_func)(int, int, int, int, int, int)　　条件設定関数