```
/*****************/
/* 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);

	// 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);

        //*******これ以降がリスト１からの変更追加分部*******

        // 各エリア内の外周が円形であることを確認します
        // エリアを囲んだ矩形の縦横の中点を円の中心座標、縦横長さの平均値を直径とします
        // 円周上に青ピクセルが存在して、円周の外側に青ピクセルが無い場合、円形の物と判断します。
        int outer_pixel = 10;           // 円の外側とする半径の差
        int margin = 5;                         // 円周上の青ピクセル数をカウントする際の半径マージン
        for (n = 0; n < area_info_counter; n++) {
                int area_w = area_info[n][7] - area_info[n][6] + 1;  //①
                int area_h = area_info[n][5] - area_info[n][4] + 1;  //②
                int diameter = (area_w + area_h) / 2;  //③
                int radius = diameter / 2;
                int center_i = (area_info[n][4] + area_info[n][5]) / 2;  //④
                int center_j = (area_info[n][6] + area_info[n][7]) / 2;  //④
                int in_blue_counter = 0;
                int in_counter_all = 0;
                int out_blue_counter = 0;
                int out_counter_all = 0;
                for (i = center_i - radius - outer_pixel; i <= center_i + radius + outer_pixel; i++) {
                        for (j = center_j - radius - outer_pixel; j <= center_j + radius + outer_pixel; j++) {
                                if (0 <= i && i < height_pixel && 0 <= j && j < width_pixel) {
                                        int circle_value = (i - center_i) * (i - center_i) + (j - center_j) * (j - center_j);
                                        if ((radius - margin) * (radius - margin) <= circle_value && circle_value <= (radius * radius)) {
                                                if (gBuff_blue[i][j] == 255) in_blue_counter++;
                                                in_counter_all++;
                                        }
                                        if ((radius + outer_pixel - margin) * (radius + outer_pixel - margin) <= circle_value &&
                                                circle_value <= (radius + outer_pixel) * (radius + outer_pixel)) {
                                                if (gBuff_blue[i][j] == 255) out_blue_counter++;
                                                out_counter_all++;
                                        }
                                }
                        }
                }
                if (0 < in_counter_all && 0 < out_counter_all) {
                        in_blue_counter = in_blue_counter * 100 / in_counter_all;  //⑤
                        out_blue_counter = out_blue_counter * 100 / out_counter_all;  //⑤
                }

                // in_blue_counterが60%以上、out_blue_counterが10%以下の場合円形とします⑥
                char circle_message[64];
                if (60 <= in_blue_counter && out_blue_counter <= 10) {
                        sprintf(circle_message, "円形です！");
                }
                else {
                        sprintf(circle_message, "円形ではありません！");
```

```
                }


                printf("  %2d  center=(%4d,%4d)  area_w=%3d  area_h=%3d  diameter=%3d
in_blue_counter=%2d  out_blue_counter=%2d  %s¥n",
                        n, center_j, center_i, area_w, area_h, diameter,
in_blue_counter, out_blue_counter, circle_message);

                // 直径の値をarea_info[][0]に、中心座標をarea_info[][2, 3]に記録します
                area_info[n][0] = diameter;
                area_info[n][2] = center_i;
                area_info[n][3] = center_j;
        }

        // 直径から何円のコインかを特定して全コインの合計額を求めます⑦
        int result_yen_all = 0;
        int diameter_th[5] = { 219, 229, 237, 246, 264 };       // 1円玉と50円玉の直径
中間、50と5円の中間・・・10と500円の中間の配列
        int yen[6] = { 1, 50, 5, 100, 10, 500 };                        // 金額の配列
        for (n = 0; n < area_info_counter; n++) {
                int k;
                int result_k = 5;
                for (k = 0; k < 5; k++) {
                        if (area_info[n][0] < diameter_th[k]) {
                                result_k = k;
                                k = 5;
                        }
                }
                result_yen_all = result_yen_all + yen[result_k];
                printf("  %2d  center=(%4d,%4d)  %3d円玉  合計金額：%4d円¥n",
                        n, area_info[n][3], area_info[n][2], yen[result_k],
result_yen_all);
        }


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

        return 0;
}
```