```
/*****************/
/* 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 LEFT_CENTER_X 754    // 左目瞳中心X座標
#define LEFT_CENTER_Y 460    // 左目瞳中心Y座標
#define RIGHT_CENTER_X 982    // 右目瞳中心X座標
#define RIGHT_CENTER_Y 450    // 右目瞳中心Y座標
#define R_MAX 114        // 処理する円エリアの半径　目の場合は（RIGHT_CENTER_X - LEFT_CENTER_X) / 2
#define EXPAND_RATIO 8      // 大きくしたい度合（推奨値：瞳の直径ピクセル数の1/8前後）

#define PAI 3.2425

// 指定座標の目を大きくします
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) {

  int i, j, n;

  int center_y[2] = { LEFT_CENTER_Y, RIGHT_CENTER_Y };
  int center_x[2] = { LEFT_CENTER_X, RIGHT_CENTER_X };
  int r_max = R_MAX;
  int ratio = EXPAND_RATIO;

  // 元画像をgBuff_xxx[][]にコピーします
  for (i = 0; i < height_pixel; i++) {
    for (j = 0; j < width_pixel; j++) {
      gBuff_red[i][j] = gRed[i][j];
      gBuff_green[i][j] = gGreen[i][j];
      gBuff_blue[i][j] = gBlue[i][j];
    }
  }

  // 設定値に従って処理します
  int r, ii, jj;
  for (n = 0; n < 2; n++) {
    int center_j = center_x[n];
    int center_i = center_y[n];

    printf("  center(%d,%d)¥n", center_j, center_i);

    // 処理範囲の外側の円から半径を1ピクセル減らしながら半径１までループします
    for (r = r_max; r >= 1; r--) {

      // 処理する円周上にコピーする元画像のピクセル座標を算出します
      for (i = center_i - r; i <= center_i + r; i++) {
        for (j = center_j - r; j <= center_j + r; j++) {
          int circle_value = (i - center_i) * (i - center_i) + (j - center_j) * (j - center_j);
          if ((r - 1) * (r - 1) <= circle_value && circle_value <= r * r) {

            // 書き込むピクセルから読み込むピクセルまでの距離をshiftとして算出します
            double shift = ratio * sin(double(r) / r_max * PAI);
            if (j < center_j) shift = -shift;

            // 処理するピクセルと中心座標を結ぶ直線が垂直な場合、読み込み先のアドレスjj, iiを算出します
            if (j == center_j) {
```

```
            jj = j;
            if (i <= center_i)  ii = i + int(shift);
            else  ii = i - int(shift);
          }

          // それ以外の場合、読み込み先のアドレスjj, iiを算出します
          else {
            double a = (double(i) - center_i) / (double(j) - center_j);
            double theta = atan(a);
            jj = j - int(shift * cos(theta));
            ii = i - int(shift * sin(theta));
          }

          // 算出したjj, iiの座標の読み込み先座標としてそのRGB値を書き込みます
          gBuff_red[i][j] = gRed[ii][jj];
          gBuff_green[i][j] = gGreen[ii][jj];
          gBuff_blue[i][j] = gBlue[ii][jj];
        }
      }
    }
  }
}


  return 0;
}
```