```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>

#define LINES 'L'
#define RECTS 'R'

using namespace std;
using namespace cv;

    /**************************************************************************
    *                           EXERCISE 4                                    *
    *                          -----------                                    *
    *                            CONTOURS                                    *
    **************************************************************************/

    /// Function Prototypes
IplImage * simpleThresholding(IplImage *, IplImage *);
IplImage * getHistogram(IplImage *, char);
IplImage * getBinImagePlus(IplImage *, unsigned);
void searchContour(IplImage *, IplImage *);
void gradSearchContour(IplImage *, IplImage *);

int main(int argc, char **argv) {

    /**************************************************************************
    *                         Gradient Method                                 *
    **************************************************************************/
    IplImage * img = cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);

    cvNamedWindow("Grey scale",0);
    cvShowImage("Grey scale", img);
    cvSaveImage("Grey_Scale.png", img);

    /// Create a binary image
    IplImage * his = getHistogram(img, LINES);

    cvNamedWindow("histogram", 0);
    cvShowImage("histogram", his);
    cvSaveImage("Histogram.png", his);

    cvWaitKey(0);

    IplImage * bin = simpleThresholding(img, his);

    cvNamedWindow("Binary Image",0);
    cvShowImage("Binary Image", bin);
    cvSaveImage("Binary_Image.png", bin);

    /// Doesn't produce a very good result and requires some priori knowledge
    /// about the scene (e.g. if one expects darker objects on a less dark
    /// background, or viceversa).
    IplImage * gradContour = cvCreateImage(cvSize(img->width - 2, img->height - 2),
IPL_DEPTH_8U, 1);
    cvSet(gradContour, 0);
    gradSearchContour(img, gradContour);

    cvNamedWindow("Gradient Method",0);
    cvShowImage("Gradient Method", gradContour);
    cvSaveImage("Gradient.png", gradContour);

    IplImage * I = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
    cvCvtColor(img, I, COLOR_GRAY2RGB);
    IplImage * m = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    cvSet(m, 0);

    cvSetImageROI(m, cvRect(1, 1, img->width - 2, img->height - 2));
    cvCopy(gradContour, m);
    cvResetImageROI(m);
    const CvMat M = cvarrToMat(m);
    cvSet(I, cvScalar(255,0,0), &M);
    cvNamedWindow("Show Gradient",0);
    cvShowImage("Show Gradient", I);
    cvSaveImage("Show_Gradient.png", I);


        /// every black pixel that has an adjacent white pixel is a border
    IplImage * binGradContour = cvCreateImage(cvSize(img->width - 2, img->height - 2),
IPL_DEPTH_8U, 1);
    cvSet(binGradContour, 0);
    searchContour(bin, binGradContour);

```

```
    cvNamedWindow("Binary-Gradient Contour",0);
    cvShowImage("Binary-Gradient Contour", binGradContour);
    cvSaveImage("Binary_Gradient_Contour.png", binGradContour);

    cvCvtColor(img, I, COLOR_GRAY2RGB);
    m = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    cvSet(m, 0);
    cvSetImageROI(m, cvRect(1,1,img->width-2,img->height-2));
    cvCopy(binGradContour, m);
    cvResetImageROI(m);
    const CvMat M_1 = cvarrToMat(m);
    cvSet(I, cvScalar(0,0,255), &M_1);
    cvNamedWindow("Show Binary Contour",0);
    cvShowImage("Show Binary Contour", I);
    cvSaveImage("Show_Binary_Contour.png", I);


    /*************************************************************************
    *                   Gradient Method - the right way                      *
    *************************************************************************/

    IplImage * Xim = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    IplImage * Yim = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    IplImage * gradientImage = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);

    float Prewitt_x [3][3] = {{ -1 , 0, 1},
                              { -1 , 0, 1},
                              { -1 , 0, 1}};
    float Prewitt_y [3][3] = {{ -1, -1, -1},
                              {  0,  0,  0},
                              {  1,  1,  1}};

    CvMat PrewittXK, PrewittYK;
    cvInitMatHeader(&PrewittXK, 3, 3, CV_32F, Prewitt_x);
    cvInitMatHeader(&PrewittYK, 3, 3, CV_32F, Prewitt_y);

    cvFilter2D(img, Xim, &PrewittXK, cvPoint(-1,-1));
    cvFilter2D(img, Yim, &PrewittYK, cvPoint(-1,-1));

    for(int i = 0; i < Xim->height; i++){
        char * px = Xim->imageData + i*Xim->widthStep;
        char * py = Yim->imageData + i*Yim->widthStep;
        char * pmag = gradientImage->imageData + i*gradientImage->widthStep;
        for(int j = 0; j < Xim->width; j++){
            float x = (float)(uchar)(*px);
            float y = (float)(uchar)(*py);
            *pmag = (uchar)sqrt(x*x + y*y);
            px++;
            py++;
            pmag++;
        }
    }

    cvNamedWindow("Prewitt", 0);
    cvShowImage("Prewitt", gradientImage);
    cvSaveImage("Prewitt.png", gradientImage);

    ////////////////////////////////////////////////////////
    float Sobel_x [3][3] = {{ -1 , 0, 1},
                            { -2 , 0, 2},
                            { -1 , 0, 1}};
    float Sobel_y [3][3] = {{ -1, -2, -1},
                            {  0,  0,  0},
                            {  1,  2,  1}};

    CvMat SobelXK, SobelYK;
    cvInitMatHeader(&SobelXK, 3, 3, CV_32F, Sobel_x);
    cvInitMatHeader(&SobelYK, 3, 3, CV_32F, Sobel_y);

    cvFilter2D(img, Xim, &SobelXK, cvPoint(-1,-1));
    cvFilter2D(img, Yim, &SobelYK, cvPoint(-1,-1));

    for(int i = 0; i < Xim->height; i++){
        char * px = Xim->imageData + i*Xim->widthStep;
        char * py = Yim->imageData + i*Yim->widthStep;
        char * pmag = gradientImage->imageData + i*gradientImage->widthStep;
        for(int j = 0; j < Xim->width; j++){
            float x = (float)(uchar)(*px);
            float y = (float)(uchar)(*py);
            *pmag = (uchar)sqrt(x*x + y*y);
            px++;
            py++;
            pmag++;
        }
```

```
    }

    cvNamedWindow("Sobel", 0);
    cvShowImage("Sobel", gradientImage);
    cvSaveImage("Sobel.png", gradientImage);

    ////////////////////////////////////////////////////////////////
    IplImage * CannyIm = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    cvLaplace(img, CannyIm, 5);
    cvNamedWindow("Laplace", 0);
    cvShowImage("Laplace", gradientImage);
    cvSaveImage("Laplace.png", gradientImage);

    ////////////////////////////////////////////////////////////////
    // Resulting image very dark
    float Roberts_x [2][2] = {{  0 , 1},
                              { -1 , 0}};
    float Roberts_y [2][2] = {{ 1,  0},
                              { 0, -1}};

    CvMat RobertsXK, RobertsYK;
    cvInitMatHeader(&RobertsXK, 2, 2, CV_32F, Roberts_x);
    cvInitMatHeader(&RobertsYK, 2, 2, CV_32F, Roberts_y);

    cvFilter2D(img, Xim, &RobertsXK, cvPoint(-1,-1));
    cvFilter2D(img, Yim, &RobertsYK, cvPoint(-1,-1));

    for(int i = 0; i < Xim->height; i++){
        char * px = Xim->imageData + i*Xim->widthStep;
        char * py = Yim->imageData + i*Yim->widthStep;
        char * pmag = gradientImage->imageData + i*gradientImage->widthStep;
        for(int j = 0; j < Xim->width; j++){
            float x = (float)(uchar)(*px);
            float y = (float)(uchar)(*py);
            *pmag = (uchar)sqrt(x*x + y*y);
            px++;
            py++;
            pmag++;
        }
    }

    cvNamedWindow("Roberts", 0);
    cvShowImage("Roberts", gradientImage);
    cvSaveImage("Roberts.png", gradientImage);


    /*************************************************************************
    *                          Laplacian Method                              *
    *************************************************************************/

    IplImage * LG_contour = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);

    /*double LG_mat [9][9] = {{0,0,1,2,2,2,1,0,0},
                            {0,1,5,10,12,10,5,1,0},
                            {1,5,15,19,16,19,15,5,1},
                            {2,10,19,-19,-64,-19,19,10,2},
                            {2,12,16,-64,-148,-64,16,12,2},
                            {2,10,19,-19,-64,-19,19,10,2},
                            {1,5,15,19,16,19,15,5,1},
                            {0,1,5,10,12,10,5,1,0},
                            {0,0,1,2,2,2,1,0,0}};*/
    /*double LG_mat [7][7] = {{0,  1,   2,  4,  2,  1,  0},
                            {1,  7,  24,  31,  24,  7,  1},
                            {2,  24,  17,  -51,  17,  24,  2},
                            {4,  31,  -51,  -248,  -51,  31,  4},
                            {2,  24,  17,  -51,  17,  24,  2},
                            {1,  7,  24,  31,  24,  7,  1},
                            {0,  1,   2,  4,  2,  1,  0}};*/
    double LG_mat [5][5] = {{1,9,19,9,1},
                            {9,58,12,58,9},
                            {19,12,-432,12,19},
                            {9,58,12,58,9},
                            {1,9,19,9,1}};
    ////////////////////////////////////////
    /*double LG_mat [3][3] = {{ 1,   4,  1},
                            { 4, -20,  4},
                            { 1,   4,  1}};*/

    CvMat LG_kernel;
    cvInitMatHeader(&LG_kernel, 5, 5, CV_64F, LG_mat);

    cvFilter2D(img, LG_contour, &LG_kernel, cvPoint(-1,-1));

    cvNamedWindow("Laplacian Contour",0);
```

```
    cvShowImage("Laplacian Contour", LG_contour);
    cvSaveImage("Laplacian_Contour.png", LG_contour);


    /********************************************************************
    *                          Contour Search                           *
    ********************************************************************/

    // To implement

    cvWaitKey(0);


    /********************************************************************
    *                             Clean-Up                              *
    ********************************************************************/
    cvReleaseImage(&img);
    cvReleaseImage(&his);
    cvReleaseImage(&bin);
    cvReleaseImage(&gradContour);
    cvReleaseImage(&binGradContour);
    cvReleaseImage(&LG_contour);

    cvDestroyAllWindows();

    return 0;
}



bool gradCheck (char *, char *, int);

void gradSearchContour(IplImage * img, IplImage * contour) {
    for(int i = 0; i < img->height - 2; i++) {
        char * p_img = img->imageData + (i + 1)*img->widthStep + 1;
        char * p_contour = contour->imageData + i*contour->widthStep;
        for(int j = 0; j < img->width - 2; j++) {
            if ( gradCheck(p_img, p_img - 1 - img->widthStep, img->widthStep) )
                    *p_contour = 255;
            p_img++;
            p_contour++;
        }
    }
}

bool gradCheck (char * c, char * p, int step) {
    for(int i = 0; i < 3; i++) {
        char * lp = p + i*step;
        for(int j = 0; j < 3; j++) {
            if (i == 1 && j == 1) {
                lp++;
                continue;
            }
            float diff = (float)((uchar)(*lp) - (uchar)(*c));
            if(diff > 23)
                return true;
            lp++;
        }
    }
    return false;
}


bool check(char *, int);

void searchContour(IplImage * bin, IplImage * contour) {
    for(int i = 0; i < bin->height - 2; i++) {
        char * p_bin = bin->imageData + (i + 1)*bin->widthStep + 1;
        char * p_contour = contour->imageData + i*contour->widthStep;
        for(int j = 0; j < bin->width - 2; j++) {
            if ( ((uchar)*p_bin == 0) && check(p_bin - 1 - bin->widthStep, bin->widthStep))
                    *p_contour = 255;
            p_bin++;
            p_contour++;
        }
    }
}

bool check (char * p, int step) {
    for(int i = 0; i < 3; i++) {
        char * lp = p + i*step;
        for(int j = 0; j < 3; j++) {
            if (i == 1 && j == 1) {
                lp++;
```

```
                continue;
            }
            if((uchar)(*lp) == 255)
                return true;
            lp++;
        }
    }
    return false;
}


    /*************************************************************************
    * This function allows the user to select a threshold and draws a new    *
    * histogram with a visual indication of the threshold. At the same time  *
    * computes the necessary numbers for the centre of mass.                 *
    *                                                                        *
    * The function needs:                                                    *
    *    - a pointer to a GREY SCALE image to apply the threshold to;        *
    *    - a pointer to a preliminary histogram image (starting point to     *
    *      draw the new histogram with the threshold);                       *
    *    - 3 pointers to int to compute the center of mass coordinates       *
    *************************************************************************/
IplImage * simpleThresholding(IplImage * G, IplImage * his) {
    unsigned int t = 0;     // Threshold
    char st = 4;            // To place correctly the threshold line in the image
    IplImage * bin;         // To contain the binary image

    printf("\nInsert Threshold: ");
    scanf("%u", &t);

    IplImage * Chis = cvCreateImage(cvGetSize(his), IPL_DEPTH_8U, 3);
        /// Represent a grey scale histogram in a 3-channel image.
        /// It will be useful later to draw the threshold on the histogram.
    cvCvtColor(his, Chis, COLOR_GRAY2RGB);
    cvFlip(Chis, Chis, 0);

    cvLine(Chis, cvPoint(t*st, 0), cvPoint(t*st, 600), cvScalar(0,0,255), 2, 4);
    cvShowImage("histogram", Chis);
    cvSaveImage("histogram.png", Chis);
    cvReleaseImage(&Chis);

    bin = getBinImagePlus(G, t);
    cvNamedWindow("Binary Image", 0);
    cvShowImage("Binary Image", bin);

    return bin;
}


    /*************************************************************************
    * From the pointer to a greyscale image, draws a histogram image and     *
    * returns a pointer to it. Two possible styles can be selected:          *
    * LINES (a line for each bin)                                            *
    * RECTS (a rectangle for each bin)                                       *
    *************************************************************************/
IplImage * getHistogram(IplImage * gray, char style) {
    int dep = 256;
    int hist[dep];  // Histogram for gray scale image
    int st = 4;     // To separate lines in the histograms, for better look ;)

        /// Initialize histograms
    for (int i = 0; i < dep; i++)
        hist[i] = 0;

        /// Calculate histogram
    for(int i = 0; i < gray->height; i++) {
        char * ptr = gray->imageData + i*gray->widthStep;
        for(int j = 0; j < gray->width; j++) {
                if(*ptr < 0) {
                    char c = *ptr;
                    uchar x = (uchar) c;
                    hist[x] += 1;
                }
                else
                    hist[(unsigned)(*ptr)] += 1;
                ptr++;
        }
    }

        /// Create image for the histograms
    IplImage * his = cvCreateImage(cvSize(st*dep,600), IPL_DEPTH_8U, 1);
    cvSet(his, 0);                      // Initialize image (all black)
    his->origin = IPL_ORIGIN_BL;        // Set the origin in the bottom left corner

```

```
        /// Draw a line/Rectangle for each bin
    if (style == LINES)
        for (int i = 0; i < dep; i++)
            if (hist[i] != 0)
                cvLine(his, cvPoint(i*st, 0), cvPoint(i*st, hist[i]/10), 150, 1, 4);
    else if (style == RECTS)
        for (int i = 0; i < dep; i++)
            if (hist[i] != 0)
                cvRectangle(his, cvPoint(i*st,hist[i]/10), cvPoint((i+1)*st,0), 150, -1,
4);
    else {
        printf("\nError. No style selected.\n");
        cvReleaseImage(&his);
        return NULL;
    }

    return his;
}


    /*************************************************************************
    * Input: a pointer to a GREY SCALE image, a threshold, 3 pointers to     *
    * integer for the necessary computation to determine the center of mass. *
    * Forms a binary image, counts the number of pixels set to 0, sums       *
    * the values of the 2 coordinates for the pixels set to 0 (used later to *
    * calculate the center of mass coordinates).                             *
    *                                                                        *
    * PS. It is possible to calculate the coordinates even in this function  *
    * but it is done later to respect the structure of the exercise          *
    *************************************************************************/
IplImage * getBinImagePlus(IplImage * img, unsigned t) {
    IplImage * I = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    for(int i = 0; i < img->height; i++) {
        char * ptr = img->imageData + i*img->widthStep;
        char * p = I->imageData + i*I->widthStep;
        for(int j = 0; j < img->width; j++) {
                if ((uchar)(*ptr) >= (uchar)t)
                    *p = 255;
                else {
                    // In this case we are on the object
                    *p = 0;
                }
            ptr++;
            p++;
        }
    }
    return I;
}

```