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


using namespace std;
using namespace cv;


void ROIs(const char *, const char *);


int main(int argc, char ** argv) {
    const char * imgName = argv[1];
    const char * binName = argv[2];
    ROIs(imgName, binName);

    return 0;
}



/// ////////////////////////////////////////////////
/// ////////////////////////////////////////////////


void ROIs(const char * imgName, const char * binName) {
    /// ////////////////////////////////////////////////////////////////////////////////
    ///                                INPUT (a binary image)                         ///
    /// ////////////////////////////////////////////////////////////////////////////////
    Mat img, bin;
    img = imread(imgName, CV_LOAD_IMAGE_GRAYSCALE);      // Binary Image
    bin = imread(binName, CV_LOAD_IMAGE_GRAYSCALE);      // Original Image

    Mat bincpy = bin.clone();                            // Copy used by 'findContour'

    blur(bincpy, bincpy, CvSize(3, 3));                  /// Improves finding of centroids but must
                                                         /// be done before thresholding.


    /// ////////////////////////////////////////////////////////////////////////////////
    ///                                FIND CONTOURS                                  ///
    /// ////////////////////////////////////////////////////////////////////////////////

    vector<vector<Point> > contours;                     // vector of vectors of Point --> Table of x and y coordinates
    vector<Vec4i> hierarchy;                             // vector of Vec4i (a 4D vector of integers) -- Not used...
```

```
49
50      findContours(bincpy, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_NONE, Point (0, 0));
51
52      /// NOTE: Probably hierarchy and the point in the last argument can be omitted.
53
54
55      /// ////////////////////////////////////////////////////////////////////////////////////
56      ///   COMPUTE MOMENTS FOR EACH CONTOURS AND EXTRACT ONLY THE NORMALIZED 1ST ORDER MOMENTS ///
57      /// ////////////////////////////////////////////////////////////////////////////////////
58
59      int N = contours.size();                                    // Easier and less confusing to use N than calling the method every time
60
61      //                      COUNT THE CONTOURS THAT HAVE ENOUGH POINTS
62
63      /// NOTE: Tests determined that 5 points is the minimum to be sure that moments can be
64      ///       computed correctly
65      int K = 0;
66      for(int i = 0; i < N; i++) {
67          if (contours[i].size() >= 5)
68              K++;
69          ///                   FOR DEBUGGING:
70          /// Display for each contour the number of points and the value of K
71          /// printf("%5d     ", contours[i].size());
72          /// printf("%d\n", K);
73      }
74
75      //                                  RECAP OF THE COUNT
76
77      cout << endl << "Total number of objects: " << N << endl;
78      cout << endl << "Number of useful objects: " << K << endl;
79
80
81      //                      FOR EACH USEFUL CONTOUR DETERMINES ALL MOMENTS
82
83      vector<Moments> mmnts(K);                                   // Vector of objects of 'Moments'
84
85      /// Keep the following 4 instructionS in mind and read the note below...
86      int j = 0;
87      for(int i = 0; i < N; i++)
88          if (contours[i].size() >= 5)
89              mmnts[j++] = moments(contours[i], false);
90
91
92      //                                NORMALIZED FIRST MOMENTS
93
94      /// NOTE: Necessary because the class 'Moments' doesn't have normalized first moments...
95      vector<Point2f> norm1stMoments(K);
96
97      for (int i = 0; i < K; i++)
98          norm1stMoments[i] = Point2f((mmnts[i].m10 / mmnts[i].m00), (mmnts[i].m01 /
```

```
            mmnts[i].m00));
 99
100
101     /// NOTE: exactly the same for cycle is repeated twice: first, to compute K, and a second time to
102     ///       calculate the moments. In the present implementation this is necessary because the value
103     ///       K is used to declare 'mmnts'.
104     ///       The most elegant (and probably correct) method is allocating dynamically 'mmnts'.
105
106
107     /// ////////////////////////////////////////////////////////////////////////////////////
108     ///                         SCAN THROUGH COORDINATES OF EACH CONTOUR                    ///
109     /// ////////////////////////////////////////////////////////////////////////////////////
110
111     // These are used to form the name of the ROI images.
112     string name = "IMG";
113     string sufix = ".png";
114     char numstr[21];
115     string result = "test";
116     int n = 0;
117
118     int X = bin.cols, Y = bin.rows;                         // Image sizes
119     int S = 10;                                             // Half side of the ROI
120
121
122     /// ///////////////   FOR DEBUGGING
123     // Show which objects have been "seen" from the original image
124     IplImage * imgcpy = cvLoadImage(imgName, CV_LOAD_IMAGE_GRAYSCALE);
125     IplImage * I = cvCreateImage(cvGetSize(imgcpy), IPL_DEPTH_8U, 3);
126     cvCvtColor(imgcpy, I, COLOR_GRAY2RGB);
127     unsigned short A = 8;
128     /// ///////////////
129
130
131     for (int i = 0; i < K; i++) {
132         // Discard ROIs that fall out of the image
133         if(norm1stMoments[i].x - S >= 0 && norm1stMoments[i].y - S >= 0 &&
              norm1stMoments[i].x + S < X && norm1stMoments[i].y + S < Y) {
134             //
                ____________________________________________________________________________
                ____________________________________________TO DO
135             /// - CHECKS: BRIGHTNESS, SATURATION, SHAPE, MULTIPLICITY
136             /// - CREATE IMAGES ONLY FOR THOSE ROIs THAT PASS THE PREVIOUS CHECKS
137
138             //
139             // Ensure a coherent numeration (i.e. Use 'n' instead of 'i' as part of the name)
140             sprintf(numstr, "%d", n);
141             result = name + numstr + sufix;
142             // Definition of ROI
143             Rect rect(norm1stMoments[i].x - S, norm1stMoments[i].y - S, 2*S, 2*S);
144             // Creation of ROI image with reasonable size
145             /// NOTE: resizing may not be allowed because it adds data to the original
```

```
146             Mat Roi;
147             resize(img(rect), Roi, Size(400,400), 1, 1, INTER_LANCZOS4);
148
149             imwrite(result, Roi);
150             Roi.release();
151
152             /// ////////////////////   FOR DEBUGGING
153             cvCircle(I, cvPoint(norm1stMoments[i].x,norm1stMoments[i].y), A, cvScalar
                  (255,0,255), 1, 4);
154             /// ////////////////////
155             n++;
156         }
157         else
158             cout << "Discarded:  " << norm1stMoments[i] << endl;
159
160
161     }
162
163     /// ////////////////////// FOR DEBUGGING
164     cvSaveImage("check.png", I);
165     cvReleaseImage(&imgcpy);
166     cvReleaseImage(&I);
167     /// ////////////////////////
168
169     bin.release();
170     img.release();
171     bincpy.release();
172
173 }
174
```