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

typedef struct THR {
    int B;
    int x;
} THR;


using namespace std;
using namespace cv;


void preprocessing (const char *);
void thresholding (const char *);
void multi(const char *);
int findBackground(int *, int);
void drawHistogram(IplImage *, int *);
IplImage * simpleThresholding(IplImage *, int);
IplImage * getBinImage(IplImage *, unsigned);


int main(int argc, char ** argv) {
    const char * imgName = argv[1];
    preprocessing(imgName);
    return 0;
}










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








void preprocessing(const char * imgName) {
    IplImage * img = cvLoadImage(imgName, CV_LOAD_IMAGE_GRAYSCALE);

    /// ////////////////////////////////////////////////////////////////////
    ///              MORPHOLOGICAL TRANSFORMATION                      ///
    /// ////////////////////////////////////////////////////////////////////

    IplImage * CL = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    cvSet(CL, 0);

    int IT = 12;

    cvMorphologyEx(img, CL, NULL, NULL, CV_MOP_OPEN, IT);
    cvErode(CL, CL, NULL, 10);
    cvDilate(CL, CL, NULL, 3);


    /// ///////////////////////////////////////////////////////////
    ///                     DISPLAY AND SAVE                    ///
    /// ///////////////////////////////////////////////////////////

    cvSaveImage("temp.png", CL);

    thresholding("temp.png");


    /// /////////////////////////////////////////////////////////
    ///                          CLEAN-UP                     ///
    /// /////////////////////////////////////////////////////////

    cvDestroyAllWindows();

    cvReleaseImage(&CL);
}











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









void thresholding(const char * imgName) {
    ///                 ORIGINAL IMAGE UPLOAD
    IplImage * img = cvLoadImage(imgName, CV_LOAD_IMAGE_GRAYSCALE);

    ///                         HISTOGRAM
    int dep = 256;
    int hist[dep];
    drawHistogram(img, hist);


    ///                 COMPUTE THE BACKGROUND = B
    int B = findBackground(hist, img->height * img->width);


    int t = 8, T;
    IplImage * bin;
    char imNum[2];
    char c;
    int i = 0;

    while(t <= 20) {
        char imName[30] = "";
        T = t + B;
        bin = simpleThresholding(img, T);
        strcat(imName, "bin");
        imNum[0] = 49 + i;
        imNum[1] = '\0';
        strcat(imName, imNum);
        strcat(imName, ".png");
        cvSaveImage(imName, bin);
        multi(imName);
        i++;
        t += 2;
    }
    ///                             CLEAN-UP
    cvReleaseImage(&img);
    cvReleaseImage(&bin);
    //cvReleaseImage(&bin1);
    //cvReleaseImage(&bin2);

}











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









void func(uchar c, int * h) {
    unsigned x = c;
    (*(h + x))++;
}

```

```cpp



void drawHistogram(IplImage * gray, int hist []) {
    int st = 4;      // To separate lines in the histograms, for better look ;)
    int dep = 256;
        /// 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++) {
                func((uchar)(*ptr), hist);
                ptr++;
        }
    }

        /// Create image for the histogram
    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

    //cvNamedWindow("histogram", 2);         // Create window to contain the image

        /// Draw the histogram - 2 different styles
    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);

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

    cvWaitKey(0);

    cvReleaseImage(&his);
}











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







```

```cpp



int findBackground(int hist[], int tot) {
    bool fmin = true, fmax = true;
    int m, M;


    for (int i = 0; fmin || fmax; i++) {
        if (fmin)
            if(hist[i]){
                m = i;
                fmin = false;
            }
        if (fmax)
            if(hist[255 - i]){
                M = 255 - i;
                fmax = false;
            }
    }


    THR res_t;
    res_t.B = m + (M - m) / 16;

    int frac = 45, cnt = 0;

    for(int i = m; i <= res_t.B; i++) {
        cnt += hist[i];
    }

    if( (res_t.x = ((cnt * 100) / tot)) == frac)
        return res_t.B;
    else if (res_t.x < frac) {
        while(res_t.x < frac) {
            res_t.B++;
            cnt += hist[res_t.B];
            res_t.x = (cnt * 100) / tot;
        }
        THR t1, t2;
        t1.B = res_t.B;
        t1.x = res_t.x;
        t2.B = res_t.B - 1;
        t2.x = ((cnt - hist[t2.B])* 100) / tot;
        if (abs(t2.x - frac) < abs(t1.x - frac) ) {
            printf("\nPercentile: %d%%    ,    Background: %d\n", t2.x, t2.B);
            return t2.B;
        }

        else {
            printf("\nPercentile: %d%%    ,    Background: %d\n", t1.x, t1.B);
            return t1.B;
        }

    }
    else { // x > frac
        while(res_t.x > frac) {
            res_t.B--;
            cnt -= hist[res_t.B];
            res_t.x = (cnt * 100) / tot;
        }
        THR t1, t2;
        t1.B = res_t.B;
        t1.x = res_t.x;
        t2.B = res_t.B + 1;
        t2.x = ((cnt + hist[t2.B])* 100) / tot;
        if (abs(t2.x - frac) < abs(t1.x - frac) ) {
            printf("\nPercentile: %d%%    ,    Background: %d\n", t2.x, t2.B);
            return t2.B;
        }
        else {
            printf("\nPercentile: %d%%    ,    Background: %d\n", t1.x, t1.B);
            return t1.B;
        }
    }
}











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








IplImage * simpleThresholding(IplImage * G, int th) {
    IplImage * bin;

    bin = getBinImage(G, th);
    cvNamedWindow("Binary Image", 0);
    cvShowImage("Binary Image", bin);

    return bin;
}








```

```cpp
IplImage * getBinImage(IplImage * img, unsigned th) {
    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)th)
                    *p = 255;
                else
                    *p = 0;
            ptr++;
            p++;
        }
    }
    return I;
}








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









void multi(const char * imgName) {
    Mat img;
    img = imread(imgName, CV_LOAD_IMAGE_GRAYSCALE);    // Already processed externally

    vector<vector<Point> > segm;
    findContours(img, segm, CV_RETR_TREE, CV_CHAIN_APPROX_NONE, Point(0, 0));
    cout << endl << "objects: "<< segm.size();

    if(segm.size() > 1) {
        cout << " -> multiple objects in the same ROI!" << endl;
    }
    img.release();
}


```