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


using namespace std;
using namespace cv;


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



/*
void drawHistogram(IplImage *, int *);
void drawHistogram(IplImage *, int *);
IplImage * getBinImage(IplImage *, unsigned);
int findContrast(IplImage *, int *, int);
*/

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




int main(int argc, char ** argv) {
    // INPUT: ROI IMAGE


    //                    FAINTNESS
    //IplImage * img;
    //img = cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
    /*
    int dep = 256;
    int hist[dep];
    int cnt = 0;
    int k = 0;

    string line;
    ifstream out(argv[1]);

    string name = "IMGR";
    string sufix = ".png";
    char numstr[21];
    string result = "test";

     while(getline(out, line)) {
```

```
59          const char * imgName = line.c_str();
60          IplImage * img = cvLoadImage(imgName, CV_LOAD_IMAGE_GRAYSCALE);
61          int hist[dep];
62          sprintf(numstr, "%d", cnt);
63          result = name + numstr + sufix;
64          IplImage * rej = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
65          cvSet(rej, 127);
66          int c = findContrast(img, hist, cnt);
67          cout << endl;
68          if(c >= 9) {
69              cvCopy(img, rej);
70              cvSaveImage(result.c_str(), rej);
71              cout << result << " is ok!" << endl;
72              cvSaveImage(result.c_str(), rej);
73          }
74          else {
75              cout << result << " is rejected!" << endl;
76              k++;
77          }
78
79          cnt++;
80          cvReleaseImage(&img);
81          cvReleaseImage(&rej);
82      }
83
84      cout << endl << k << " images out of " << cnt << " have been rejected." << endl;
85      */
86
87      //                    MULTIPLICITY
88      //                     PROCESSING:
89      // No filtering
90      // Opening: 12 iterations
91      // Erosion: 10 iterations
92      // Dilation: 1 iteration
93      // 2 distinct thresholding: t1 = 20 & t2 = 11
94
95      int dep = 256;
96      int hist[dep];
97      int cnt = 0;
98      int k = 0;
99
100     string line;
101     ifstream out(argv[1]);
102
103     string name = "IMGM";
104     string sufix = ".png";
105     char numstr[21];
106     string result = "test";
107
108     while(getline(out, line)) {
109         const char * imgName = line.c_str();
110
111         sprintf(numstr, "%d", cnt);
112         result = name + numstr + sufix;
113         cout << endl;
114
115         IplImage * img = cvLoadImage(imgName, CV_LOAD_IMAGE_GRAYSCALE);
116         IplImage * cpy = cvCloneImage(img);
```

```
117
118         // Opening: 12 iterations
119         cvMorphologyEx(cpy, cpy, NULL, NULL, CV_MOP_OPEN, 12);
120         // Erosion: 10 iterations
121         cvErode(cpy, cpy, NULL, 10);
122         // Dilation: 3 iterations
123         cvDilate(cpy, cpy, NULL, 3);
124         // 2 distinct thresholding: t1 = 20 & t2 = 11
125
126         IplImage * cpyLow = cvCloneImage(cpy);
127
128         calcHistogram(cpy, hist);
129         int B = findBackground(hist, img->height * img->width);
130         int t1 = 20;
131         int T1 = t1 + B;
132         IplImage * bin1 = simpleThresholding(cpy, T1);
133
134         if(multi(bin1, hist)) {
135             cout << imgName << " is rejected!" << endl;
136             cvReleaseImage(&cpy);
137             cvReleaseImage(&cpyLow);
138             cvReleaseImage(&bin1);
139             cnt++;
140             k++;
141             continue;
142         }
143
144         calcHistogram(cpyLow, hist);
145         B = findBackground(hist, img->height * img->width);
146         int t2 = 11;
147         int T2 = t2 + B;
148         IplImage * bin2 = simpleThresholding(cpyLow, T2);
149
150         if(multi(bin2, hist)) {
151             cout << imgName << " is rejected!" << endl;
152             cvReleaseImage(&cpy);
153             cvReleaseImage(&cpyLow);
154             cvReleaseImage(&bin1);
155             cvReleaseImage(&bin2);
156             cnt++;
157             k++;
158             continue;
159         }
160
161         cvSaveImage(result.c_str(), img);
162         cout << imgName << " is ok!" << endl;
163
164         cnt++;
165         cvReleaseImage(&img);
166         cvReleaseImage(&cpy);
167         cvReleaseImage(&cpyLow);
168         cvReleaseImage(&bin1);
169         cvReleaseImage(&bin2);
170
171     }
172
173     cout << endl << k << " images out of " << cnt << " have been rejected." << endl;
174
```

```
175
176     // THIRD TEST: SATURATION
177
178
179     // FOURTH TEST: SHAPE
180
181
182     // SAVE
183
184
185     // CLEAN
186
187     return 0;
188 }
189
190
191
192
193 /// ////////////////////////////////////////////////////////////////////////////////////////// ↵
      //////////
194 /// ////////////////////////////////////////////////////////////////////////////////////////// ↵
      //////////
195
196
197
198
199 bool multi(IplImage * img, int * hist) {
200     vector<vector<Point> > segm;
201     Mat imgM = cvarrToMat(img);
202     findContours(imgM, segm, CV_RETR_TREE, CV_CHAIN_APPROX_NONE, Point(0, 0));
203     imgM.release();
204
205     if(segm.size() > 1)
206         return true;
207     else
208         return false;
209 }
210
211
212
213
214 /// ////////////////////////////////////////////////////////////////////////////////////////// ↵
      //////////
215 /// ////////////////////////////////////////////////////////////////////////////////////////// ↵
      //////////
216
217
218
219
220 int findBackground(int hist[], int tot) {
221     bool fmin = true, fmax = true;
222     int m, M;
223
224     for (int i = 0; fmin || fmax; i++) {
225         if (fmin)
226             if(hist[i]){
227                 m = i;
228                 fmin = false;
```

```
229             }
230         if (fmax)
231             if(hist[255 - i]){
232                 M = 255 - i;
233                 fmax = false;
234             }
235     }
236     THR res_t;
237     res_t.B = m + (M - m) / 16;
238 
239     int frac = 45, cnt = 0;
240 
241     for(int i = m; i <= res_t.B; i++)
242         cnt += hist[i];
243 
244     if( (res_t.x = ((cnt * 100) / tot)) == frac)
245         return res_t.B;
246     else if (res_t.x < frac) {
247         while(res_t.x < frac) {
248             res_t.B++;
249             cnt += hist[res_t.B];
250             res_t.x = (cnt * 100) / tot;
251         }
252         THR t1, t2;
253         t1.B = res_t.B;
254         t1.x = res_t.x;
255         t2.B = res_t.B - 1;
256         t2.x = ((cnt - hist[t2.B])* 100) / tot;
257         if (abs(t2.x - frac) < abs(t1.x - frac) )
258             return t2.B;
259         else
260             return t1.B;
261     }
262     else { // x > frac
263         while(res_t.x > frac) {
264             res_t.B--;
265             cnt -= hist[res_t.B];
266             res_t.x = (cnt * 100) / tot;
267         }
268         THR t1, t2;
269         t1.B = res_t.B;
270         t1.x = res_t.x;
271         t2.B = res_t.B + 1;
272         t2.x = ((cnt + hist[t2.B])* 100) / tot;
273         if (abs(t2.x - frac) < abs(t1.x - frac) )
274             return t2.B;
275         else
276             return t1.B;
277     }
278 }
279 
280 
281 
282 
283 /// ////////////////////////////////////////////////////////////////////////////////// ↵
      ///////////
284 /// ////////////////////////////////////////////////////////////////////////////////// ↵
      ///////////
```

```
285
286
287
288
289 IplImage * simpleThresholding(IplImage * G, int th) {
290     IplImage * bin;
291
292     bin = getBinImage(G, th);
293
294     return bin;
295 }
296
297
298
299
300
301 IplImage * getBinImage(IplImage * img, unsigned th) {
302     IplImage * I = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
303     for(int i = 0; i < img->height; i++) {
304         char * ptr = img->imageData + i*img->widthStep;
305         char * p = I->imageData + i*I->widthStep;
306         for(int j = 0; j < img->width; j++) {
307                 if ((uchar)(*ptr) >= (uchar)th)
308                     *p = 255;
309                 else
310                     *p = 0;
311             ptr++;
312             p++;
313         }
314     }
315     return I;
316 }
317
318
319
320
321 /// ////////////////////////////////////////////////////////////////////////////////// ↵
      ///////////
322 /// ////////////////////////////////////////////////////////////////////////////////// ↵
      ///////////
323
324
325
326
327 void func(uchar c, int * h) {
328     unsigned x = c;
329     (*(h + x))++;
330 }
331
332
333 void calcHistogram(IplImage * gray, int * hist) {
334     int dep = 256;
335     for (int i = 0; i < dep; i++)
336         hist[i] = 0;
337
338         /// Calculate histogram
339     for(int i = 0; i < gray->height; i++) {
340         char * ptr = gray->imageData + i*gray->widthStep;
```

```
341         for(int j = 0; j < gray->width; j++) {
342                 func((uchar)(*ptr), hist);
343                 ptr++;
344         }
345     }
346 }
347
348
349
350
351
352 /// ////////////////////////////////////////////////////////////////////////////////////// ↵
      ///////////
353 /// ////////////////////////////////////////////////////////////////////////////////////// ↵
      ///////////
354
355
356
357
358
359 int findContrast(IplImage * gray, int hist [], int l) {
360     int st = 4;      // To separate lines in the histograms, for better look ;)
361     int dep = 256;
362         /// Initialize histograms
363     for (int i = 0; i < dep; i++)
364         hist[i] = 0;
365
366         /// Calculate histogram
367     for(int i = 0; i < gray->height; i++) {
368         char * ptr = gray->imageData + i*gray->widthStep;
369         for(int j = 0; j < gray->width; j++) {
370                 func((uchar)(*ptr), hist);
371                 ptr++;
372         }
373     }
374
375         /// Create image for the histogram
376     IplImage * his = cvCreateImage(cvSize(st*dep,600), IPL_DEPTH_8U, 1);
377     cvSet(his, 0);                  // Initialize image (all black)
378     his->origin = IPL_ORIGIN_BL;    // Set the origin in the bottom left corner
379
380
381     for (int i = 0; i < dep; i++)
382         if (hist[i] != 0)
383             cvLine(his, cvPoint(i*st, 0), cvPoint(i*st, hist[i]/10), 150, 1, 4);
384
385     //cvSaveImage("Histogram.png", his);
386
387     int cnt = 0, TOT = (gray->height * gray->width);
388     int p5, p95;
389     bool f1 = true, f2 = true;
390     for(int i = 0; i < 256; i++) {
391         cnt += hist[i];
392         if(f1 && ((cnt * 100) / TOT) >= 5) {
393             p5 = i;
394             f1 = false;
395             cout << endl;
396             cout << ((cnt * 100) / TOT) << " percentile: " << i << endl;
```

```
397         }
398 
399         if(f2 && ((cnt * 100) / TOT) >= 95) {
400             p95 = i;
401             f2 = false;
402             cout << ((cnt * 100) / TOT) << " percentile: " << i << endl;
403         }
404 
405     }
406 
407     cout << "contrast: " << (p95 - p5) << endl;
408     cout << endl;
409 
410     return (p95 - p5);
411 }
412 
413 
```