```cpp
// FaceRec9.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "omp.h"
#include "math.h"
#include "fftw3.h"
#include "new.h"
#include <list>
#include <chrono>
#include <thread>
#include <opencv2/core/core.hpp>
#include "opencv2/imgcodecs/imgcodecs.hpp"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include "GetFreq_REC.h"
#include "FaceRec9.h"

using namespace cv;
using namespace std;

typedef unsigned char byte;

#define MAX_LOADSTRING 100

byte* bytes, * bytes1, * bytes3, * bytesX;

char POM[200], POM1[200], POM2[200];
int selClan;
int N1 = 768, N2 = 1366, N = 768;
int PROLAZ, PROLAZ1, pprolaz, povrat = 0, KAMERA = 1;
int brUz, brUzObj;
```

```
34  int brObjects = 7, brFreq = 1014;  // brFreq_REC = 338*3 (RGB)
35  int poz, poz1, cancel;
36  LRESULT checkBoxes[2];
37  int XX, YY;
38  static HWND hwndButton;
39
40  double largest, largest1;
41  double S_R[768][1366], S_G[768][1366], S_B[768][1366];
42
43  double** largX;
44
45  double** A_R, ** A_G, ** A_B;
46
47  double*** R_REC;
48
49  GetFreq_REC** REC;
50
51  char** names;
52
53  FILE* stream;
54
55  VideoCapture cap;
56
57  //  ********************************************************************************
58  //  *****   Get Image functions   ***************************************************
59  //  ********************************************************************************
60
61  //  *****   Recognition function  ***************************************
62
63  void GetImage_FACEREC(HWND hWnd) {
64
65      namedWindow("FaceRecognition", WINDOW_NORMAL);
66      setWindowProperty("FaceRecognition", WND_PROP_FULLSCREEN, WINDOW_FULLSCREEN);
```

```
67
68
69      int i, j;
70
71      Mat frame = Mat(768, 1366, CV_8UC3);
72
73      while (1) {
74
75          Mat temp;
76
77          bool bSuccess = cap.read(temp); // read new frame from video
78
79          if (!bSuccess) // if not success, break loop
80          {
81              destroyAllWindows();
82              MessageBox(hWnd, TEXT("Frame not captured !"), TEXT("FaceRec9 -> Message:"), MB_OK);
83              break;
84          }
85
86          //  ***  cut frame from large temp image
87
88          {
89              Rect ROI(469, 264, 1366, 768);  //  temp image: 1296x2304 pix
90              temp(ROI).copyTo(frame);  //  frame image: 768x1366 pix
91          }
92
93          //  ***  put it to the buffer for processing in recognition subroutine
94
95          std::memcpy(bytes, frame.data, 3147264 * sizeof(byte));
96
97          //  ***  draw 768x768 pix rectangle in the frame
98
99          rectangle(frame, Point(299, 0), Point(1067, 768), Scalar(63, 191, 127, 255), 11, LINE_AA, 0);
```

```
100
101         int R = 0; int G = 255; int B = 0;
102
103         //  ***  draw circles arround eyes
104
105         RotatedRect rRect = RotatedRect(Point2f(491, 216), Size2f(192, 192), 0);  // 491
106         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
107
108         rRect = RotatedRect(Point2f(491, 216), Size2f(96, 96), 0);
109         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
110
111         rRect = RotatedRect(Point2f(875, 216), Size2f(192, 192), 0);  //  875
112         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
113
114         rRect = RotatedRect(Point2f(875, 216), Size2f(96, 96), 0);
115         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
116
117         if (largest > 0.5) {
118             rectangle(frame, Point(18, 458), Point(118, 558), Scalar(127, 127, 127, 255), FILLED, LINE_AA, 0);
119
120             Mat frame_REC = Mat(96, 96, CV_8UC3, bytes1).clone();
121
122             {
123                 Rect ROI(20, 460, 96, 96);
124                 frame_REC.copyTo(frame(ROI));
125             }
126         }
127
128         if (largest > 0.5) {
129             G = 0; R = 255;
130         }
131
132         //  ***  draw circles with changed colours
```

```
133
134         rRect = RotatedRect(Point2f(491, 216), Size2f(192, 192), 0);  // 491
135         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
136
137         rRect = RotatedRect(Point2f(491, 216), Size2f(13, 13), 0);
138         ellipse(frame, rRect, Scalar(B, G, R, 255), FILLED, LINE_AA);
139
140         rRect = RotatedRect(Point2f(875, 216), Size2f(192, 192), 0);  //  875
141         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
142
143         rRect = RotatedRect(Point2f(875, 216), Size2f(13, 13), 0);
144         ellipse(frame, rRect, Scalar(B, G, R, 255), FILLED, LINE_AA);
145
146         //  ***  draw names and probabilities for detected face
147
148         i = poz / brUzObj;
149         sprintf_s(POM, 200, "%s", names[i]);
150         j = 640 - ((int)strlen(names[i]) / 2 * 40) + 20;
151         if (largest > 0.5)
152             putText(frame, POM, Point2f((float)j, 100), FONT_HERSHEY_TRIPLEX, 2, Scalar(0, 0, 255, 255), 1,
              LINE_AA, false);
153         else putText(frame, POM, Point2f((float)j, 100), FONT_HERSHEY_TRIPLEX, 2, Scalar(255, 0, 0, 255), 1,
            LINE_AA, false);
154
155         sprintf_s(POM, 200, "imageNr = %d", poz + 1);
156         if (largest > 0.5)
157             putText(frame, POM, Point2f(430, 680), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(0, 0, 255, 255), 1,
              LINE_AA, false);
158         else putText(frame, POM, Point2f(430, 680), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(255, 0, 0, 255), 1,
            LINE_AA, false);
159
160         sprintf_s(POM, 200, "prob = %.5f", largest);
161         if (largest > 0.5)
```

```
162             putText(frame, POM, Point2f(430, 700), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(0, 0, 255, 255), 1,
                  LINE_AA, false);
163         else putText(frame, POM, Point2f(430, 700), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(255, 0, 0, 255), 1,
              LINE_AA, false);
164
165         imshow("FaceRecognition", frame);
166
167         if ((waitKey(1) == 27) || (KAMERA == 1)) //wait for 'esc' key press for 1ms. If 'esc' key is pressed,
              break loop
168         {
169             destroyAllWindows();
170             break;
171         }
172     }
173
174     KAMERA = 1;
175 }
176
177 //  *****   Recognition with Face Detection   ******************
178
179 void GetImage_FACEDET_REC(HWND hWnd) {
180
181     int i, j;
182
183     namedWindow("FaceDetRec", WINDOW_NORMAL);
184     moveWindow("FaceDetRec", XX, YY);
185     setWindowProperty("FaceDetRec", WND_PROP_FULLSCREEN, WINDOW_FULLSCREEN);
186
187     Mat frame = Mat(768, 1366, CV_8UC3);
188
189     while (1) {
190
191         Mat temp;
```

```
192
193         bool bSuccess = cap.read(temp); // read new frame from video
194
195         if (!bSuccess) //if not success, break loop
196         {
197             destroyAllWindows();
198             MessageBox(hWnd, TEXT("Frame not captured !"), TEXT("FaceRec9 -> Message:"), MB_OK);
199             break;
200         }
201
202         //  ***  cut 768x1366 pix frame from 2304x1296 pix video image
203
204         {
205             Rect ROI(469, 264, 1366, 768);  //  temp image: (2304, 1296)
206             temp(ROI).copyTo(frame);
207         }
208
209         //  ***  buffer it
210
211         std::memcpy(bytes, frame.data, 3147264 * sizeof(byte));
212
213         //  ***  draw rectangles
214
215         rectangle(frame, Point(299, 0), Point(1067, 768), Scalar(63, 127, 191, 255), 11, LINE_AA, 0);
216
217         //  ***  if above threashold show image from detection process
218
219         if (largest > 0.2) {
220
221             rectangle(frame, Point(48, 248), Point(104, 304), Scalar(127, 127, 127, 255), FILLED, LINE_AA, 0);
222
223             Mat frame_REC = Mat(52, 52, CV_8UC3, bytes1).clone();
224
```

```
225             {
226                 Rect ROI(50, 250, 52, 52);
227                 frame_REC.copyTo(frame(ROI));
228             }
229
230             Mat frame1_REC = Mat(96, 96, CV_8UC3, bytes3).clone();
231
232             rectangle(frame, Point(118, 443), Point(218, 543), Scalar(127, 127, 127, 255), FILLED, LINE_AA, 0);
233
234             {
235                 Rect ROI(120, 445, 96, 96);
236                 frame1_REC.copyTo(frame(ROI));
237             }
238
239             putText(frame, "DB Image:", Point2f(120, 435), FONT_HERSHEY_PLAIN, 1, Scalar(255, 0, 0, 255), 1,
                  LINE_AA, false);
240         }
241
242         //  ***  if above threashold show image from recognition process
243
244         if (largest1 > 0.5) {
245
246             Mat frame1_REC = Mat(96, 96, CV_8UC3, bytes3).clone();
247
248             rectangle(frame, Point(118, 443), Point(218, 543), Scalar(127, 127, 127, 255), FILLED, LINE_AA, 0);
249
250             {
251                 Rect ROI(120, 445, 96, 96);
252                 frame1_REC.copyTo(frame(ROI));
253             }
254
255             putText(frame, "DB Image:", Point2f(120, 435), FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255, 255), 1,
                  LINE_AA, false);
```

```
256         }
257
258         //  ***  detection pass nr. counter
259
260         sprintf_s(POM, 200, "Pass: %d", PROLAZ);
261         putText(frame, POM, Point2f(20, 150), FONT_HERSHEY_PLAIN, 1.5, Scalar(0, 0, 255, 255), 1, LINE_AA, false);
262
263         sprintf_s(POM, 200, "PassLocal: %d", PROLAZ1);
264         putText(frame, POM, Point2f(50, 220), FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255, 255), 1, LINE_AA, false);
265
266         //  *** display name, probability and image nr. of recognized face
267
268         i = poz1 / brUzObj;
269         sprintf_s(POM, 200, "%s", names[i]);
270         j = 640 - ((int)strlen(names[i]) / 2 * 40) + 20;
271         if (largest1 > 0.5)
272             putText(frame, POM, Point2f((float)j, 100), FONT_HERSHEY_TRIPLEX, 2, Scalar(0, 0, 255, 255), 1,
                  LINE_AA, false);
273         else putText(frame, POM, Point2f((float)j, 100), FONT_HERSHEY_TRIPLEX, 2, Scalar(255, 0, 0, 255), 1,
              LINE_AA, false);
274
275         sprintf_s(POM, 200, "Face pattern image:");
276         putText(frame, POM, Point2f(20, 190), FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255, 255), 1, LINE_AA, false);
277
278         sprintf_s(POM, 200, "imageNr = %d", poz1 + 1);
279         if (largest1 > 0.5)
280             putText(frame, POM, Point2f(530, 680), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(0, 0, 255, 255), 1,
                  LINE_AA, false);
281         else putText(frame, POM, Point2f(530, 680), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(255, 0, 0, 255), 1,
              LINE_AA, false);
282
283         sprintf_s(POM, 200, "prob = %.5f", largest1);
284         if (largest1 > 0.5)
```

```
285             putText(frame, POM, Point2f(530, 700), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(0, 0, 255, 255), 1,
                  LINE_AA, false);
286         else putText(frame, POM, Point2f(530, 700), FONT_HERSHEY_COMPLEX_SMALL, 1, Scalar(255, 0, 0, 255), 1,
              LINE_AA, false);
287
288         imshow("FaceDetRec", frame);
289
290         if ((waitKey(1) == 27) || (KAMERA == 1)) //wait for 'esc' key press for 1ms. If 'esc' key is pressed,
              break loop
291         {
292             destroyAllWindows();
293             break;
294         }
295
296     }
297
298     KAMERA = 1;
299 }
300
301 //  *****   Database Creation   *****
302
303 void GetImage_RecogDB(HWND hWnd) {
304
305     namedWindow("RecognitionDB", WINDOW_NORMAL);
306     setWindowProperty("RecognitionDB", WND_PROP_FULLSCREEN, WINDOW_FULLSCREEN);
307
308     int R = 0; int G = 0; int B = 255;
309     Mat frame = Mat(768, 1366, CV_8UC3);
310
311     while (1) {
312
313         Mat temp;
314
```

```
315         bool bSuccess = cap.read(temp); // read a new frame from video
316
317         if (!bSuccess) //if not success, break loop
318         {
319             destroyAllWindows();
320             MessageBox(hWnd, TEXT("Frame not captured !"), TEXT("FaceRec9 -> Message:"), MB_OK);
321             break;
322         }
323
324         {
325             Rect ROI(469, 264, 1366, 768);  //  temp image: (2304x1296)
326             temp(ROI).copyTo(frame);
327         }
328
329         std::memcpy(bytes, frame.data, 3147264 * sizeof(byte));  //  1366x768x3
330
331         if (PROLAZ == 50) {
332             R = 255; B = 0;
333         }
334
335         if (PROLAZ == 350) {
336             G = 255; R = 0;
337         }
338
339         RotatedRect rRect = RotatedRect(Point2f(491, 216), Size2f(192, 192), 0);  //  240
340         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
341
342         rRect = RotatedRect(Point2f(875, 216), Size2f(192, 192), 0);
343         ellipse(frame, rRect, Scalar(B, G, R, 255), 1, LINE_AA);
344
345         rRect = RotatedRect(Point2f(491, 216), Size2f(15, 15), 0);
346         ellipse(frame, rRect, Scalar(B, G, R, 255), FILLED, LINE_AA);
347
```

```
348         rRect = RotatedRect(Point2f(875, 216), Size2f(15, 15), 0);
349         ellipse(frame, rRect, Scalar(B, G, R, 255), FILLED, LINE_AA);
350
351         imshow("RecognitionDB", frame);
352
353         if ((waitKey(1) == 27) || (KAMERA == 1)) //wait for 'esc' key press for 1ms. If 'esc' key is pressed,
              break loop
354         {
355             destroyAllWindows();
356             break;
357         }
358     }
359     KAMERA = 1;
360 }
361
362 //  ******************************************
363 //  ***   DYNAMIC FIELDS   *********************
364 //  ******************************************
365
366 int CreateDynamicFields() {
367
368     char ch;
369     int i, j, k;
370
371     if (fopen_s(&stream, ".\\brUzoraka.txt", "r") == 0) {
372
373         i = 0;
374         while ((ch = fgetc(stream)) != ',')
375             POM[i++] = ch;
376         POM[i] = '\0';
377         brUz = atoi(POM);
378
379         i = 0;
```

```
380         while ((ch = fgetc(stream)) != '#')
381             POM[i++] = ch;
382         POM[i] = '\0';
383         brUzObj = atoi(POM);
384         fclose(stream);
385     }
386     else return (0);
387
388     if (fopen_s(&stream, ".\\monitors.dat", "r") == 0) {
389         fread(checkBoxes, sizeof(LRESULT), (2), stream);
390         fclose(stream);
391
392         if (checkBoxes[1] == BST_CHECKED) {
393             XX = GetSystemMetrics(SM_CXSCREEN);
394             YY = -GetSystemMetrics(SM_CYSCREEN);
395         }
396         else {
397             XX = 0;
398             YY = 0;
399         }
400     }
401     else return (0);
402
403     REC = new GetFreq_REC * [brObjects];
404
405     if (brUz == 0) return (1);
406
407     R_REC = new double** [brObjects];
408
409     for (int i = 0; i < brObjects; i++) {
410         R_REC[i] = new double* [brUz];
411         for (int j = 0; j < brUz; j++)
412             R_REC[i][j] = new double[brFreq];
```

```
413     }
414
415     largX = new double* [brObjects];
416     for (i = 0; i < brObjects; i++)
417         largX[i] = new double[brUz];
418
419     names = new char* [brUz / brUzObj];
420
421     if ((fopen_s(&stream, ".\\names.txt", "r") == 0) && (stream != NULL)) {
422
423         j = 0; ch = '$';
424         while (ch != '#') {
425
426             i = 0;
427             while (((ch = fgetc(stream)) != '\n') && (ch != '#')) {
428                 POM[i++] = ch;
429             }
430             POM[i++] = '\0';
431
432             if (ch == '#') break;
433
434             names[j] = new char[i];
435
436             for (k = 0; k < i; k++) names[j][k] = POM[k];
437
438             j++;
439         }
440
441         fclose(stream);
442     }
443     else return(0);
444
445     return(1);
```

```
446 }
447
448 void DestroyDynamicFields() {
449
450     int i, j;
451
452     if (brUz == 0) {
453         delete[] REC;
454         return;
455     }
456
457     for (i = 0; i < brUz / brUzObj; i++)
458         delete[] names[i];
459     delete[] names;
460
461     for (i = 0; i < brObjects; i++) {
462         for (j = 0; j < brUz; j++)
463             delete[] R_REC[i][j];
464         delete[] R_REC[i];
465     }
466     delete[] R_REC;
467
468     for (i = 0; i < brObjects; i++)
469         delete[] largX[i];
470     delete[] largX;
471
472     delete[] REC;
473 }
474
475 int SpremiPodatke(int SW) {
476
477     int i;
478
```

```
479     if ((fopen_s(&stream, ".\\brUzoraka.txt", "w+t") == 0) && (stream != NULL)) {
480         fprintf_s(stream, "%d,%d#", brUz, brUzObj);
481         fclose(stream);
482     }
483     else return(0);
484
485     if ((fopen_s(&stream, ".\\names.txt", "w+t") == 0) && (stream != NULL)) {
486         if (SW == 0) {
487             for (i = 0; i < (brUz - brUzObj) / brUzObj; i++)
488                 fprintf_s(stream, "%s\n", names[i]);
489             fprintf_s(stream, "%d. %s\n", i + 1, POM);
490         }
491         else {
492             for (i = 0; i < (brUz) / brUzObj; i++)
493                 fprintf_s(stream, "%s\n", names[i]);
494         }
495         fprintf_s(stream, "#");
496         fclose(stream);
497     }
498     else return(0);
499     return(1);
500 }
501
502 //   ************************************************
503 //   ***   reduce x time x image to y time y image   ***
504 //   ************************************************
505
506 void Reduce_Image_FFTW(int x, int y) {
507
508     int i, j;
509
510     double* x_R = (double*)fftw_malloc(sizeof(double) * (size_t)x * (size_t)x);
511     for (i = 0; i < x; i++)
```

```
512         for (j = 0; j < x; j++)
513             x_R[i * x + j] = S_R[i][j];
514     fftw_complex* y_R = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) * (size_t)x * ((size_t)x / 2 + 1));
515     fftw_plan plan_R = fftw_plan_dft_r2c_2d(x, x, x_R, y_R, FFTW_ESTIMATE);
516
517     double* x_G = (double*)fftw_malloc(sizeof(double) * x * x);
518     for (i = 0; i < x; i++)
519         for (j = 0; j < x; j++)
520             x_G[i * x + j] = S_G[i][j];
521     fftw_complex* y_G = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) * (size_t)x * ((size_t)x / 2 + 1));
522     fftw_plan plan_G = fftw_plan_dft_r2c_2d(x, x, x_G, y_G, FFTW_ESTIMATE);
523
524     double* x_B = (double*)fftw_malloc(sizeof(double) * (size_t)x * (size_t)x);
525     for (i = 0; i < x; i++)
526         for (j = 0; j < x; j++)
527             x_B[i * x + j] = S_B[i][j];
528     fftw_complex* y_B = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) * (size_t)x * ((size_t)x / 2 + 1));
529     fftw_plan plan_B = fftw_plan_dft_r2c_2d(x, x, x_B, y_B, FFTW_ESTIMATE);
530
531     fftw_execute(plan_R);
532     fftw_execute(plan_G);
533     fftw_execute(plan_B);
534
535     for (i = 0; i < y / 2; i++)
536         for (j = 0; j < y / 2; j++) {
537             y_R[i * (y / 2 + 1) + j][0] = y_R[i * (x / 2 + 1) + j][0];
538             y_R[i * (y / 2 + 1) + j][1] = y_R[i * (x / 2 + 1) + j][1];
539
540             y_G[i * (y / 2 + 1) + j][0] = y_G[i * (x / 2 + 1) + j][0];
541             y_G[i * (y / 2 + 1) + j][1] = y_G[i * (x / 2 + 1) + j][1];
542
543             y_B[i * (y / 2 + 1) + j][0] = y_B[i * (x / 2 + 1) + j][0];
544             y_B[i * (y / 2 + 1) + j][1] = y_B[i * (x / 2 + 1) + j][1];
```

```
545         }
546
547     for (i = y / 2; i < y; i++)
548         for (j = 0; j < y / 2; j++) {
549             y_R[i * (y / 2 + 1) + j][0] = y_R[(i + (x - y)) * (x / 2 + 1) + j][0];
550             y_R[i * (y / 2 + 1) + j][1] = y_R[(i + (x - y)) * (x / 2 + 1) + j][1];
551
552             y_G[i * (y / 2 + 1) + j][0] = y_G[(i + (x - y)) * (x / 2 + 1) + j][0];
553             y_G[i * (y / 2 + 1) + j][1] = y_G[(i + (x - y)) * (x / 2 + 1) + j][1];
554
555             y_B[i * (y / 2 + 1) + j][0] = y_B[(i + (x - y)) * (x / 2 + 1) + j][0];
556             y_B[i * (y / 2 + 1) + j][1] = y_B[(i + (x - y)) * (x / 2 + 1) + j][1];
557         }
558
559     fftw_plan plan1_R = fftw_plan_dft_c2r_2d(y, y, y_R, x_R, FFTW_ESTIMATE);
560     fftw_plan plan1_G = fftw_plan_dft_c2r_2d(y, y, y_G, x_G, FFTW_ESTIMATE);
561     fftw_plan plan1_B = fftw_plan_dft_c2r_2d(y, y, y_B, x_B, FFTW_ESTIMATE);
562
563     fftw_execute(plan1_R);
564     fftw_execute(plan1_G);
565     fftw_execute(plan1_B);
566
567     for (i = 0; i < y; i++)
568         for (j = 0; j < y; j++) {
569             S_R[i][j] = x_R[i * y + j] / ((size_t)x * x);
570             if (S_R[i][j] < 0) S_R[i][j] = 0;
571             else if (S_R[i][j] > 255) S_R[i][j] = 255;
572
573             S_G[i][j] = x_G[i * y + j] / ((size_t)x * x);
574             if (S_G[i][j] < 0) S_G[i][j] = 0;
575             else if (S_G[i][j] > 255) S_G[i][j] = 255;
576
577             S_B[i][j] = x_B[i * y + j] / ((size_t)x * x);
```

```
578             if (S_B[i][j] < 0) S_B[i][j] = 0;
579             else if (S_B[i][j] > 255) S_B[i][j] = 255;
580         }
581
582     fftw_free(x_R); fftw_free(x_G); fftw_free(x_B);
583     fftw_free(y_R); fftw_free(y_G); fftw_free(y_B);
584     fftw_destroy_plan(plan_R); fftw_destroy_plan(plan_G); fftw_destroy_plan(plan_B);
585     fftw_destroy_plan(plan1_R); fftw_destroy_plan(plan1_G); fftw_destroy_plan(plan1_B);
586 }
587
588 void GetFaceRegion() {
589
590     int i, j, k, m, n, p, r, s;
591     double pom, pom1;
592
593     //  ***  resize NxN (768x768) pix image to 80x80 pix
594
595     Reduce_Image_FFTW(N, 80);
596
597     //  ************
598     //  ***  1   ***  buffer image (80x80 pix)
599     //  ************
600
601     for (i = 0; i < 80; i++)
602         for (j = 0; j < 80; j++) {
603             A_B[i][j] = S_B[i][j];
604             A_G[i][j] = S_G[i][j];
605             A_R[i][j] = S_R[i][j];
606         }
607
608     //  ***  resize original image to 52x52 pix  ***
609
610     Reduce_Image_FFTW(80, 52);
```

```
611
612     //  ***  buffer 52x52 pix image
613
614     for (i = 0; i < 52; i++)
615         for (j = 0; j < 52; j++) {
616             bytesX[(size_t)i * 156 + (size_t)j * 3] = (byte)S_B[i][j];
617             bytesX[(size_t)i * 156 + (size_t)j * 3 + 1] = (byte)S_G[i][j];
618             bytesX[(size_t)i * 156 + (size_t)j * 3 + 2] = (byte)S_R[i][j];
619         }
620
621     //   ***   initialize objects
622
623     k = 0;
624     for (m = 0; m < 14; m += 13)
625         for (n = 0; n < 27; n += 13) {
626             for (p = 0; p < 26; p++)
627                 for (r = 0; r < 26; r++) {
628                     REC[k]->SX_R[p][r] = S_R[m + p][n + r];
629                     REC[k]->SX_G[p][r] = S_G[m + p][n + r];
630                     REC[k]->SX_B[p][r] = S_B[m + p][n + r];
631                 }
632             k++;
633         }
634
635     m = 26; n = 13;
636     for (p = 0; p < 26; p++)
637         for (r = 0; r < 26; r++) {
638             REC[k]->SX_R[p][r] = S_R[m + p][n + r];
639             REC[k]->SX_G[p][r] = S_G[m + p][n + r];
640             REC[k]->SX_B[p][r] = S_B[m + p][n + r];
641         }
642
643     //  ***  process objects
```

```
644
645     #pragma omp parallel sections num_threads(2)
646     {
647         #pragma omp section
648         {
649             REC[0]->GetFreq();
650             REC[1]->GetFreq();
651             REC[2]->GetFreq();
652         }
653
654         #pragma omp section
655         {
656             REC[3]->GetFreq();
657             REC[4]->GetFreq();
658             REC[5]->GetFreq();
659         }
660     }
661
662     REC[6]->GetFreq();
663
664     //  ***  calculate probabilities
665
666     for (i = 0; i < brUz; i++) {
667         pom = 0;
668         for (j = 0; j < brObjects; j++)
669             pom += largX[j][i];
670         pom /= brObjects;
671
672         if (pom > largest) {
673             largest = pom;
674             poz = i;
675         }
676     }
```

```
677
678     //  ***********
679     //  ***  2  ***  cut frames from original image A   ***
680     //  ***********
681
682     s = 76;
683     while ((s > 51) && (KAMERA == 0)) {
684         for (m = 0; m < 80 - s; m++)
685             for (n = 0; n < 80 - s; n++) {
686                 for (i = m; i < m + s; i++)
687                     for (j = n; j < n + s; j++) {
688                         S_B[i - m][j - n] = A_B[i][j];
689                         S_G[i - m][j - n] = A_G[i][j];
690                         S_R[i - m][j - n] = A_R[i][j];
691                     }
692
693                     //  ***  resize to 52x52 pix
694
695                 if (s > 52) Reduce_Image_FFTW(s, 52);
696
697                 k = 0;
698                 for (i = 0; i < 14; i += 13)
699                     for (j = 0; j < 27; j += 13) {
700                         for (p = 0; p < 26; p++)
701                             for (r = 0; r < 26; r++) {
702                                 REC[k]->SX_R[p][r] = S_R[i + p][j + r];
703                                 REC[k]->SX_G[p][r] = S_G[i + p][j + r];
704                                 REC[k]->SX_B[p][r] = S_B[i + p][j + r];
705                             }
706                         k++;
707                     }
708
709                 i = 26; j = 13;
```

```
710                 for (p = 0; p < 26; p++)
711                     for (r = 0; r < 26; r++) {
712                         REC[k]->SX_R[p][r] = S_R[i + p][j + r];
713                         REC[k]->SX_G[p][r] = S_G[i + p][j + r];
714                         REC[k]->SX_B[p][r] = S_B[i + p][j + r];
715                     }
716
717                     //  ***  process objects
718
719                 #pragma omp parallel sections num_threads(2)
720                 {
721                     #pragma omp section
722                     {
723                         REC[0]->GetFreq();
724                         REC[1]->GetFreq();
725                         REC[2]->GetFreq();
726                     }
727
728                     #pragma omp section
729                     {
730                         REC[3]->GetFreq();
731                         REC[4]->GetFreq();
732                         REC[5]->GetFreq();
733                     }
734                 }
735
736                 REC[6]->GetFreq();
737
738                 //  ***  calculate probabilities
739
740                 pom1 = largest;
741                 for (i = 0; i < brUz; i++) {
742                     pom = 0;
```

```
743                     for (j = 0; j < brObjects; j++)
744                         pom += largX[j][i];
745                     pom /= brObjects;
746 
747                     if (pom > largest) {
748                         largest = pom;
749                         poz = i;
750                     }
751                 }
752 
753                 // ***  buffer image
754 
755                 if (largest > pom1) {
756                     for (i = 0; i < 52; i++)
757                         for (j = 0; j < 52; j++) {
758                             bytesX[(size_t)i * 156 + (size_t)j * 3] = (byte)S_B[i][j];
759                             bytesX[(size_t)i * 156 + (size_t)j * 3 + 1] = (byte)S_G[i][j];
760                             bytesX[(size_t)i * 156 + (size_t)j * 3 + 2] = (byte)S_R[i][j];
761                         }
762                 }
763             }
764         s -= 4;
765     }
766 }
767 
768 void Distances(double** G, int* put) {
769 
770     std::list<int> N, path;
771     std::list<int>::iterator n, m, it, it1;
772     int i;
773 
774     for (i = 0; i < 300; ++i)
775         N.push_back(i);
```

```
776
777     n = N.begin();
778     path.splice(path.end(), N, n);
779
780     while (!N.empty()) {
781
782         it1 = N.begin(); it = path.begin();
783         for (m = path.begin(); m != path.end(); ++m) {
784             for (n = N.begin(); n != N.end(); ++n) {
785                 if (G[*n][*m] > G[*it1][*it]) {
786                     it = m; it1 = n;
787                 }
788             }
789         }
790
791         path.splice(path.end(), N, it1);
792     }
793
794     i = 0;
795     for (n = path.begin(); n != path.end(); ++n)
796         put[i++] = *n;
797 }
798
799 void RecognizedFace(int rez) {
800
801     int i, j;
802
803     sprintf_s(POM1, 200, ".\\MEMBERS\\%d.bmp", rez);
804     byte* bytes5 = new byte[96 * 96 * 3];
805     Mat frame = imread(POM1, IMREAD_COLOR);
806     if (!frame.empty()) {
807         std::memcpy(bytes5, frame.data, ((size_t)96 * 96 * 3) * sizeof(byte));
808         for (i = 0; i < 96; i++)
```

```
809             for (j = 0; j < 96; j++) {
810                 S_B[i][j] = bytes5[i * 288 + j * 3];
811                 S_G[i][j] = bytes5[i * 288 + j * 3 + 1];
812                 S_R[i][j] = bytes5[i * 288 + j * 3 + 2];
813             }
814     }
815     delete[]bytes5;
816 }
817
818 void ShowMemberImage(HWND hwnd, int selClan1) {
819
820     int i, j;
821
822     sprintf_s(POM, 200, "%s", names[selClan1]);
823     i = 0;
824     while (POM[i] != ' ') i++;
825     j = 0;
826     while (POM[++i] != '\0') POM[j++] = POM[i];
827     POM[j] = '\0';
828     sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\%d.bmp", POM, (selClan1 * 300 + 1));
829
830     bytes = new byte[96 * 96 * 3];
831     Mat frame = imread(POM1, IMREAD_COLOR);
832     if (!frame.empty()) {
833         std::memcpy(bytes, frame.data, ((size_t)96 * 96 * 3) * sizeof(byte));
834
835         for (i = 0; i < 96; i++)
836             for (j = 0; j < 96; j++) {
837                 S_B[i][j] = bytes[i * 288 + j * 3];
838                 S_G[i][j] = bytes[i * 288 + j * 3 + 1];
839                 S_R[i][j] = bytes[i * 288 + j * 3 + 2];
840             }
841
```

```
842         HDC hdc = GetDC(hwnd);
843         if (selClan1 > -1) {
844             for (i = 0; i < 96; i++)
845                 for (j = 0; j < 96; j++)
846                     SetPixel(hdc, j + 414, i, RGB((int)(S_R[i][j]), (int)(S_G[i][j]), (int)(S_B[i][j])));
847         }
848         ReleaseDC(hwnd, hdc);
849     }
850
851     delete[]bytes;
852 }
853
854 void ChangeButtonFont(HWND hWnd) {
855     LOGFONT lf;
856     HFONT font;
857
858     memset(&lf, 0, sizeof(LOGFONT));
859     lf.lfHeight = 48;
860     lf.lfWidth = 12;
861     lstrcpy(lf.lfFaceName, _T("Monotype Corsiva"));
862     font = CreateFontIndirect(&lf);
863
864     SendMessage(hWnd, WM_SETFONT, (WPARAM)font, TRUE);
865 }
866
867 // Global Variables:
868 HINSTANCE hInst;                                // current instance
869 WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
870 WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
871
872 // Forward declarations of functions included in this code module:
873 ATOM                MyRegisterClass(HINSTANCE hInstance);
874 BOOL                InitInstance(HINSTANCE, int);
```

```
875  LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
876  INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
877  INT_PTR CALLBACK    UnesiIme(HWND, UINT, WPARAM, LPARAM);
878  INT_PTR CALLBACK    PregledClanovaBaze(HWND, UINT, WPARAM, LPARAM);
879  INT_PTR CALLBACK    IzbaciClanBaze(HWND, UINT, WPARAM, LPARAM);
880  INT_PTR CALLBACK    Create30ElementsDB(HWND, UINT, WPARAM, LPARAM);
881  INT_PTR CALLBACK    DBImagesResize(HWND, UINT, WPARAM, LPARAM);
882  INT_PTR CALLBACK    CheckBoxes(HWND, UINT, WPARAM, LPARAM);
883
884  int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
885      _In_opt_ HINSTANCE hPrevInstance,
886      _In_ LPWSTR    lpCmdLine,
887      _In_ int       nCmdShow)
888  {
889      UNREFERENCED_PARAMETER(hPrevInstance);
890      UNREFERENCED_PARAMETER(lpCmdLine);
891
892      // TODO: Place code here.
893
894      // Initialize global strings
895      LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
896      LoadStringW(hInstance, IDC_FACEREC9, szWindowClass, MAX_LOADSTRING);
897      MyRegisterClass(hInstance);
898
899      // Perform application initialization:
900      if (!InitInstance(hInstance, nCmdShow))
901      {
902          return FALSE;
903      }
904
905      HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_FACEREC9));
906
907      MSG msg;
```

```
908
909     // Main message loop:
910     while (GetMessage(&msg, nullptr, 0, 0))
911     {
912         if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
913         {
914             TranslateMessage(&msg);
915             DispatchMessage(&msg);
916         }
917     }
918
919     return (int)msg.wParam;
920 }
921
922 ATOM MyRegisterClass(HINSTANCE hInstance)
923 {
924     WNDCLASSEXW wcex;
925
926     wcex.cbSize = sizeof(WNDCLASSEX);
927
928     wcex.style = CS_HREDRAW | CS_VREDRAW;
929     wcex.lpfnWndProc = WndProc;
930     wcex.cbClsExtra = 0;
931     wcex.cbWndExtra = 0;
932     wcex.hInstance = hInstance;
933     wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_FACEREC9));
934     wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
935     wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 3);
936     wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_FACEREC9);
937     wcex.lpszClassName = szWindowClass;
938     wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
939
940     return RegisterClassExW(&wcex);
```

```
941 }
942
943 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
944 {
945     hInst = hInstance; // Store instance handle in our global variable
946
947     HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPED | WS_SYSMENU | WS_MINIMIZEBOX,
948         CW_USEDEFAULT, 0, 468 + 16, 90 + 59, nullptr, nullptr, hInstance, nullptr);
949
950     if (!hWnd)
951     {
952         return FALSE;
953     }
954
955     ShowWindow(hWnd, nCmdShow);
956     UpdateWindow(hWnd);
957
958     return TRUE;
959 }
960
961 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
962 {
963     int i, j, k, m, n, p, r;
964     double pom;
965
966     switch (message)
967     {
968     case WM_CREATE:
969     {
970         povrat = CreateDynamicFields();
971         if ((povrat == 1) && (brUz == 1)) brUz = 0;
972         pprolaz = 1;
973
```

```
974         hwndButton = CreateWindowW(TEXT("button"),
975             TEXT(""), WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_BITMAP | BS_PUSHBUTTON,
976             310, 17, 140, 60, hWnd, NULL, ((LPCREATESTRUCT)lParam)->hInstance, NULL);
977
978         //      ChangeButtonFont(hwndButton);
979
980         HANDLE hImg = LoadImage(NULL, L"face.bmp", IMAGE_BITMAP, 0, 0, LR_DEFAULTCOLOR | LR_DEFAULTSIZE |
              LR_LOADFROMFILE);
981         SendMessage(hwndButton, BM_SETIMAGE, IMAGE_BITMAP, (LPARAM)hImg);
982
983         //      ShowWindow(hwndButton, SW_HIDE);
984     }
985     break;
986
987     case WM_KILLFOCUS:
988     {
989         if (hwndButton == (HWND)wParam) {
990             SetFocus(hWnd);
991
992             HDC hdc = GetDC(hWnd);
993             HGDIOBJ penGreen = CreatePen(NULL, 1, RGB(223, 63, 63));
994             SelectObject(hdc, GetStockObject(NULL_BRUSH));
995             SelectObject(hdc, penGreen);
996             Rectangle(hdc, 308, 15, 452, 79);
997             ReleaseDC(hWnd, hdc);
998
999             SendMessage(hWnd, WM_COMMAND, ID_RECOGNITION_RECOGNITIONWITHFACEDETECTION, 1L);
1000        }
1001
1002        InvalidateRect(hWnd, NULL, TRUE);
1003    }
1004    break;
1005
```

```
1006        case WM_COMMAND:
1007        {
1008            int wmId = LOWORD(wParam);
1009            // Parse the menu selections:
1010            switch (wmId)
1011            {
1012                //   ************************************************************
1013                //   *****     RECOGNITION     **********************************
1014                //   ************************************************************
1015
1016            case ID_RECOGNITION_RECOGNITION:
1017            {
1018                if (KAMERA == 0) break;
1019                KAMERA = 0;
1020
1021                if (brUz == 0)
1022                {
1023                    MessageBox(hWnd, TEXT("Recognition database is empty !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1024                    KAMERA = 1;
1025                    break;
1026                }
1027
1028                cap.open("rtsp://admin:Oknardaj1@192.168.1.11/cam/realmonitor?channel=1&subtype=0");
1029                if (!cap.isOpened())  // if not success, exit program
1030                {
1031                    MessageBox(hWnd, TEXT("Camera not connected !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1032                    KAMERA = 1;
1033                    break;
1034                }
1035
1036                DisableProcessWindowsGhosting();
1037
1038                bytes = new byte[(size_t)N1 * (size_t)N2 * 3]();  //  buffer for captured image 768x1366 pix RGB
```

```
1039            bytes1 = new byte[27648]();  //  96x96x3
1040
1041            //  ***  create and initialize recognition object structure
1042
1043            for (i = 0; i < brObjects; i++) {
1044                REC[i] = new GetFreq_REC();
1045                REC[i]->brUz = brUz;
1046                REC[i]->sWitch = 1;
1047                REC[i]->largest = largX[i];
1048                REC[i]->R = R_REC[i];
1049            }
1050
1051            //  ***  load recognition DB data
1052
1053            for (j = 0; j < brUz; j++) {
1054                sprintf_s(POM, 200, ".\\DAT_DB_REC\\%d.dat", j + 1);
1055                if (fopen_s(&stream, POM, "r+b") == 0) {
1056                    for (i = 0; i < brObjects; i++) {
1057                        fread(R_REC[i][j], sizeof(double), (brFreq), stream);
1058                    }
1059                    fclose(stream);
1060                }
1061                else {
1062                    MessageBox(hWnd, TEXT("Recognition DB not loaded !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1063                    goto LAB1;
1064                }
1065            }
1066
1067            omp_set_nested(1);
1068            omp_set_dynamic(0);
1069
1070            #pragma omp parallel sections shared(bytes, bytes1, largest, poz, KAMERA) num_threads(2)
1071            {
```

```
1072                #pragma omp section
1073                {
1074                    GetImage_FACEREC(hWnd);  //  image capture function
1075                }
1076
1077                #pragma omp section
1078                {
1079                    DisableProcessWindowsGhosting();
1080
1081                    std::this_thread::sleep_for(std::chrono::milliseconds(200));  // time delay
1082
1083                    while (KAMERA == 0) {
1084
1085                        //  ***  split captured image (768x1366 pix) to RGB components
1086
1087                        for (i = 0; i < N1; i++)
1088                            for (j = 0; j < N2; j++) {
1089                                S_B[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3];
1090                                S_G[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3 + 1];
1091                                S_R[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3 + 2];
1092                            }
1093
1094                        //  ***  cut 768x768 pix image for each RGB component
1095
1096                        for (i = 0; i < N; i++)
1097                            for (j = 0; j < N; j++)
1098                                S_R[i][j] = S_R[i + (N1 - N) / 2][j + (N2 - N) / 2];
1099
1100                        for (i = 0; i < N; i++)
1101                            for (j = 0; j < N; j++)
1102                                S_G[i][j] = S_G[i + (N1 - N) / 2][j + (N2 - N) / 2];
1103
1104                        for (i = 0; i < N; i++)
```

```
1105                            for (j = 0; j < N; j++)
1106                                S_B[i][j] = S_B[i + (N1 - N) / 2][j + (N2 - N) / 2];
1107
1108                        //  ***  resize RGB components to 40x40 pix
1109
1110                        Reduce_Image_FFTW(N, 96);
1111
1112                        for (i = 0; i < 96; i++)
1113                            for (j = 0; j < 96; j++) {
1114                                bytes1[i * 288 + 3 * j] = (byte)S_B[i][j];
1115                                bytes1[i * 288 + 3 * j + 1] = (byte)S_G[i][j];
1116                                bytes1[i * 288 + 3 * j + 2] = (byte)S_R[i][j];
1117                            }
1118
1119                        Reduce_Image_FFTW(96, 52);
1120
1121                        //  ***  load data to object
1122
1123                        k = 0;
1124                        for (m = 0; m < 14; m += 13)
1125                            for (n = 0; n < 27; n += 13) {
1126                                for (p = 0; p < 26; p++)
1127                                    for (r = 0; r < 26; r++) {
1128                                        REC[k]->SX_R[p][r] = S_R[m + p][n + r];
1129                                        REC[k]->SX_G[p][r] = S_G[m + p][n + r];
1130                                        REC[k]->SX_B[p][r] = S_B[m + p][n + r];
1131                                    }
1132                                k++;
1133                            }
1134
1135                        m = 26; n = 13;
1136                        for (p = 0; p < 26; p++)
1137                            for (r = 0; r < 26; r++) {
```

```
1138                                    REC[k]->SX_R[p][r] = S_R[m + p][n + r];
1139                                    REC[k]->SX_G[p][r] = S_G[m + p][n + r];
1140                                    REC[k]->SX_B[p][r] = S_B[m + p][n + r];
1141                                }
1142
1143                            //  ***  process objects
1144
1145                            #pragma omp parallel sections num_threads(2)
1146                            {
1147                                #pragma omp section
1148                                {
1149                                    REC[0]->GetFreq();
1150                                    REC[1]->GetFreq();
1151                                    REC[2]->GetFreq();
1152                                }
1153
1154                                #pragma omp section
1155                                {
1156                                    REC[3]->GetFreq();
1157                                    REC[4]->GetFreq();
1158                                    REC[5]->GetFreq();
1159                                }
1160                            }
1161
1162                            REC[6]->GetFreq();
1163
1164                            largest = 0; poz = -1;
1165                            for (i = 0; i < brUz; i++) {
1166                                pom = 0;
1167                                for (j = 0; j < brObjects; j++)
1168                                    pom += largX[j][i];
1169                                pom /= brObjects;
1170
```

```
1171                                if (pom > largest) {
1172                                    largest = pom;
1173                                    poz = i;
1174                                }
1175                            }
1176                        } // end while()
1177
1178                    } // inner pragma
1179                } // outer pragma
1180
1181  LAB1:         KAMERA = 1;
1182                delete[] bytes;
1183                delete[] bytes1;
1184
1185                fftw_cleanup();
1186
1187                cap.release();
1188            }
1189
1190            break;
1191
1192            //   ************************************************************
1193            //   *****     RECOGNITION WITH FACEDETECTION    ****************
1194            //   ************************************************************
1195
1196            case ID_RECOGNITION_RECOGNITIONWITHFACEDETECTION:
1197            {
1198
1199                if (KAMERA == 0) break;
1200                KAMERA = 0;
1201
1202                if (brUz == 0)
1203                {
```

```
1204                MessageBox(hWnd, TEXT("Recognition database is empty !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1205                KAMERA = 1;
1206                break;
1207            }
1208
1209            cap.open("rtsp://admin:Oknardaj1@192.168.1.11/cam/realmonitor?channel=1&subtype=0");
1210            if (!cap.isOpened())  // if not success, exit program
1211            {
1212                MessageBox(hWnd, TEXT("Camera not connected !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1213                KAMERA = 1;
1214                break;
1215            }
1216
1217            DisableProcessWindowsGhosting();
1218
1219            bytes = new byte[(size_t)N1 * (size_t)N2 * 3]();  //  image captured from camera (N1 = 768, N2 = 1366)
1220            bytes1 = new byte[8112]();  //  52x52x3
1221            bytes3 = new byte[27648]();  // 96x96x3
1222
1223            A_R = new double* [80];
1224            A_G = new double* [80];
1225            A_B = new double* [80];
1226
1227            for (i = 0; i < 80; i++) {
1228                A_R[i] = new double[80]();
1229                A_G[i] = new double[80]();
1230                A_B[i] = new double[80]();
1231            }
1232
1233            //  ***  recognition objects creation and initialization
1234
1235            for (i = 0; i < brObjects; i++) {
1236                REC[i] = new GetFreq_REC();  //  REC[i] = pointer to recognition object program structure
```

```
1237                REC[i]->brUz = brUz;
1238                REC[i]->sWitch = 1;
1239                REC[i]->largest = largX[i];
1240                REC[i]->R = R_REC[i];
1241            }
1242
1243            //  ***  load face rec. DB
1244
1245            for (j = 0; j < brUz; j++) {
1246
1247                sprintf_s(POM, 200, ".\\DAT_DB_REC\\%d.dat", j + 1);
1248                if (fopen_s(&stream, POM, "r+b") == 0) {
1249                    for (i = 0; i < brObjects; i++) {
1250                        fread(R_REC[i][j], sizeof(double), (brFreq), stream);
1251                    }
1252                    fclose(stream);
1253                }
1254                else {
1255                    MessageBox(hWnd, TEXT("FaceRecognition DB not loaded !"), TEXT("FaceRec9 -> Message:"),
                      MB_OK);
1256                    KAMERA = 1;
1257                    goto LAB2;
1258                }
1259            }
1260
1261            largest1 = largest = 0; poz = poz1 = -1;
1262            PROLAZ = 0;
1263
1264            omp_set_nested(1);
1265            omp_set_dynamic(0);
1266
1267            #pragma omp parallel sections shared(bytes, bytes1, bytes3, largest, KAMERA, \
1268            largest1, poz, poz1, PROLAZ, PROLAZ1) num_threads(2)
```

```
1269                {
1270                    #pragma omp section
1271                    {
1272                        GetImage_FACEDET_REC(hWnd);  //  image capture function
1273                    }
1274
1275                    #pragma omp section
1276                    {
1277                        DisableProcessWindowsGhosting();
1278
1279                        std::this_thread::sleep_for(std::chrono::milliseconds(200));  // time delay
1280
1281                        while (KAMERA == 0) {
1282
1283                            largest1 = largest = 0; poz1 = poz = -1; PROLAZ1 = 0;
1284
1285                            for (p = 0; p < 52; p++)
1286                                for (r = 0; r < 52; r++) {
1287                                    bytes1[p * 156 + r * 3] = 0;
1288                                    bytes1[p * 156 + r * 3 + 1] = 0;
1289                                    bytes1[p * 156 + r * 3 + 2] = 0;
1290                                }
1291                            for (i = 0; i < 96; i++)
1292                                for (j = 0; j < 96; j++) {
1293                                    bytes3[i * 288 + j * 3] = 0;
1294                                    bytes3[i * 288 + j * 3 + 1] = 0;
1295                                    bytes3[i * 288 + j * 3 + 2] = 0;
1296                                }
1297
1298                            bytesX = new byte[8112];
1299
1300                            while ((largest < 0.2) && (KAMERA == 0)) {
1301
```

```
1302                            //  ***  split captured image to RGB components
1303
1304                            for (i = 0; i < N1; i++)
1305                                for (j = 0; j < N2; j++) {
1306                                    S_B[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3];
1307                                    S_G[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3 + 1];
1308                                    S_R[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3 + 2];
1309                                }
1310
1311                            //  ***  cut part of the image inside rectangle for each RGB component
1312
1313                            for (i = 0; i < N; i++)
1314                                for (j = 0; j < N; j++)
1315                                    S_R[i][j] = S_R[i + (N1 - N) / 2][j + (N2 - N) / 2];
1316
1317                            for (i = 0; i < N; i++)
1318                                for (j = 0; j < N; j++)
1319                                    S_G[i][j] = S_G[i + (N1 - N) / 2][j + (N2 - N) / 2];
1320
1321                            for (i = 0; i < N; i++)
1322                                for (j = 0; j < N; j++)
1323                                    S_B[i][j] = S_B[i + (N1 - N) / 2][j + (N2 - N) / 2];
1324
1325                            //  ***  split image to regions for faster face detection and recognition
1326
1327                            GetFaceRegion();
1328
1329                            PROLAZ1++;
1330                        }
1331
1332                        if (KAMERA == 1) {
1333                            delete[]bytesX;
1334                            break;
```

```
1335                        }
1336
1337                        for (i = 0; i < 52; i++)
1338                            for (j = 0; j < 52; j++) {
1339                                bytes1[i * 156 + j * 3] = bytesX[i * 156 + j * 3];
1340                                bytes1[i * 156 + j * 3 + 1] = bytesX[i * 156 + j * 3 + 1];
1341                                bytes1[i * 156 + j * 3 + 2] = bytesX[i * 156 + j * 3 + 2];
1342                            }
1343
1344                        delete[]bytesX;
1345
1346                        //  ***  buffer DB image for display
1347
1348                        largest1 = largest;  poz1 = poz;
1349                        if (poz1 > -1)
1350                            RecognizedFace(poz + 1);
1351
1352                        for (i = 0; i < 96; i++)
1353                            for (j = 0; j < 96; j++) {
1354                                bytes3[i * 288 + j * 3] = (byte)S_B[i][j];
1355                                bytes3[i * 288 + j * 3 + 1] = (byte)S_G[i][j];
1356                                bytes3[i * 288 + j * 3 + 2] = (byte)S_R[i][j];
1357                            }
1358
1359                        std::this_thread::sleep_for(std::chrono::milliseconds(6000));  // time delay
1360
1361                        PROLAZ++;
1362
1363                    } // end while()
1364
1365                } // end pragma inner
1366            } // end pragma outer
1367
```

```
1368 LAB2:       for (i = 0; i < 80; i++) {
1369                 delete[] A_R[i];
1370                 delete[] A_G[i];
1371                 delete[] A_B[i];
1372             }
1373             delete[] A_R; delete[] A_G; delete[] A_B;
1374
1375             delete[]bytes3;
1376             delete[]bytes1;
1377             delete[]bytes;
1378
1379             fftw_cleanup();
1380
1381             cap.release();
1382         }
1383
1384         break;
1385
1386         //   ****************************************************************************
1387         //   *****     ADD NEW RECOGNITION AND DETECTION DB MEMBER (DISK)     *************
1388         //   ****************************************************************************
1389
1390         case ID_RECOGNITIONDB_ADDNEWRECOGNITIONDBMEMBER:
1391         {
1392             if (KAMERA == 0) break;
1393             KAMERA = 0;
1394
1395             cap.open("rtsp://admin:Oknardaj1@192.168.1.11/cam/realmonitor?channel=1&subtype=0", CAP_ANY);
1396             if (!cap.isOpened())  // if not success, exit program
1397             {
1398                 MessageBox(hWnd, TEXT("Camera not connected !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1399                 KAMERA = 1;
1400                 break;
```

```
1401            }
1402
1403            DisableProcessWindowsGhosting();
1404
1405            bytes = new byte[(size_t)N1 * (size_t)N2 * 3]();  //  ***  buffer for captured image from camera (N1 =>
                   768, N2 = 1366)
1406
1407            //  ***  initialize recognition and detection objects (Class)
1408
1409            for (i = 0; i < brObjects; i++) {
1410                REC[i] = new GetFreq_REC();
1411                REC[i]->brUz = brUz;
1412                REC[i]->sWitch = 0;
1413            }
1414
1415            omp_set_nested(1);
1416            omp_set_dynamic(0);
1417
1418            #pragma omp parallel sections shared(bytes, KAMERA, PROLAZ) num_threads(2)
1419            {
1420                #pragma omp section
1421                {
1422                    GetImage_RecogDB(hWnd);  // ***  procedure for image capture from camera
1423                }
1424
1425                #pragma omp section
1426                {
1427                    DisableProcessWindowsGhosting();
1428
1429                    std::this_thread::sleep_for(std::chrono::milliseconds(200));  //  time delay
1430
1431                    //  ***  images saving loop
1432
```

```
1433                        PROLAZ = 0;  //  counter
1434                        while ((PROLAZ < (50 + 300)) && (KAMERA == 0)) {  //  ***  brUzObj = nr. of images per member =
                              (300)
1435
1436                            //  ***  split captured image to RGB component arrays
1437
1438                            for (i = 0; i < N1; i++)
1439                                for (j = 0; j < N2; j++) {
1440                                    S_B[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3];
1441                                    S_G[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3 + 1];
1442                                    S_R[i][j] = bytes[(size_t)i * ((size_t)N2 * 3) + (size_t)j * 3 + 2];
1443                                }
1444
1445                            //  ***  cut image inside white rectangle (768x768 pix, N = 768) for each component
1446
1447                            for (i = 0; i < N; i++)
1448                                for (j = 0; j < N; j++)
1449                                    S_R[i][j] = S_R[i + (N1 - N) / 2][j + (N2 - N) / 2];
1450
1451                            for (i = 0; i < N; i++)
1452                                for (j = 0; j < N; j++)
1453                                    S_G[i][j] = S_G[i + (N1 - N) / 2][j + (N2 - N) / 2];
1454
1455                            for (i = 0; i < N; i++)
1456                                for (j = 0; j < N; j++)
1457                                    S_B[i][j] = S_B[i + (N1 - N) / 2][j + (N2 - N) / 2];
1458
1459                            //  ***  create dir for new member
1460
1461                            if (PROLAZ == 49) {
1462                                CreateDirectoryA(".\\MEMBERS\\NEW", NULL);
1463                            }
1464
```

```
1465                        //  ***  skip first 50 images for head positioning before saving
1466
1467                        if (PROLAZ > 49) {
1468
1469                            Mat frame2;
1470
1471                            bytesX = new byte[27648];  //  96x96x3
1472
1473                            //  ***  resize image, put image in the buffer and save with imwrite()
1474
1475                            Reduce_Image_FFTW(N, 96);
1476
1477                            for (i = 0; i < 96; i++)
1478                                for (j = 0; j < 96; j++) {
1479                                    bytesX[i * 288 + 3 * j] = (byte)S_B[i][j];
1480                                    bytesX[i * 288 + 3 * j + 1] = (byte)S_G[i][j];
1481                                    bytesX[i * 288 + 3 * j + 2] = (byte)S_R[i][j];
1482                                }
1483
1484                            sprintf_s(POM, 200, ".\\MEMBERS\\NEW\\%d.bmp", brUz / brUzObj * 300 + PROLAZ - 49);
1485                            frame2 = Mat(96, 96, CV_8UC3, bytesX).clone();
1486                            imwrite(POM, frame2);
1487
1488                            delete[] bytesX;
1489                        }  //  end while()
1490
1491                        //  ***  time delays
1492
1493                        if (PROLAZ < 50) std::this_thread::sleep_for(std::chrono::milliseconds(300));
1494                        else std::this_thread::sleep_for(std::chrono::milliseconds(2));
1495
1496                        PROLAZ++;
1497
```

```
1498                    }  // end while()
1499
1500                    if ((PROLAZ > 48) && (KAMERA == 1)) {
1501                        system("rmdir /Q /S .\\MEMBERS\\NEW");
1502                        goto LAB3;
1503                    }
1504
1505                    if (PROLAZ == (50 + 300)) {
1506
1507                        //  ***  give name to the new member directory
1508
1509 LABX:                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, UnesiIme);
1510                        if (cancel == 1) {
1511                            system("rmdir /Q /S .\\MEMBERS\\NEW");
1512                            goto LAB3;
1513                        }
1514
1515                        if ((POM[0] == '\0') || (POM[0] == ' ')) {
1516                            MessageBox(hWnd, TEXT("Wrong input, try again !"), TEXT("FaceRec9 -> Message:"),
                                 MB_OK);
1517                            goto LABX;
1518                        }
1519
1520                        sprintf_s(POM1, 200, ".\\MEMBERS\\%s", POM);
1521                        sprintf_s(POM2, 200, ".\\MEMBERS\\NEW");
1522                        if (rename(POM2, POM1) != 0) {
1523                            MessageBox(hWnd, TEXT("This name already exists, type new name !"), TEXT("FaceRec9 ->
                                 Message:"), MB_OK);
1524                            goto LABX;
1525                        }
1526
1527                        sprintf_s(POM2, 200, "%s\\DB_REC", POM1);
1528                        CreateDirectoryA(POM2, NULL);
```

```
1529
1530                        sprintf_s(POM2, 200, "%s\\Members30", POM1);
1531                        CreateDirectoryA(POM2, NULL);
1532
1533                        //  ***  databases creation loop
1534
1535                        PROLAZ = 0;
1536                        while ((PROLAZ < 300) && (KAMERA == 0)) {
1537
1538                            Mat frame;
1539
1540                            bytesX = new byte[27648];  // 96x96x3
1541
1542                            //  ***  load image
1543
1544                            sprintf_s(POM2, 200, "%s\\%d.bmp", POM1, (brUz / brUzObj * 300 + PROLAZ + 1));
1545                            frame = imread(POM2, IMREAD_COLOR);
1546                            if (frame.empty()) {
1547                                PROLAZ = -4;
1548                                goto LAB3;
1549                            }
1550
1551                            //  ***  put image in the buffer
1552
1553                            size_t size = frame.total() * frame.elemSize();
1554                            std::memcpy(bytesX, frame.data, size * sizeof(byte));
1555
1556                            //  ***  split image into RGB component arrays
1557
1558                            for (i = 0; i < 96; i++)
1559                                for (j = 0; j < 96; j++) {
1560                                    S_B[i][j] = bytesX[i * 288 + j * 3];
1561                                    S_G[i][j] = bytesX[i * 288 + j * 3 + 1];
```

```
1562                                    S_R[i][j] = bytesX[i * 288 + j * 3 + 2];
1563                                }
1564
1565                            delete[] bytesX;
1566
1567                            //  ***  resize image
1568
1569                            Reduce_Image_FFTW(96, 52);
1570
1571                            //  ***  initialize objects
1572
1573                            k = 0;
1574                            for (m = 0; m < 14; m += 13)
1575                                for (n = 0; n < 27; n += 13) {
1576                                    for (p = 0; p < 26; p++)
1577                                        for (r = 0; r < 26; r++) {
1578                                            REC[k]->SX_R[p][r] = S_R[m + p][n + r];
1579                                            REC[k]->SX_G[p][r] = S_G[m + p][n + r];
1580                                            REC[k]->SX_B[p][r] = S_B[m + p][n + r];
1581                                        }
1582                                    k++;
1583                                }
1584
1585                            m = 26; n = 13;
1586                            for (p = 0; p < 26; p++)
1587                                for (r = 0; r < 26; r++) {
1588                                    REC[k]->SX_R[p][r] = S_R[m + p][n + r];
1589                                    REC[k]->SX_G[p][r] = S_G[m + p][n + r];
1590                                    REC[k]->SX_B[p][r] = S_B[m + p][n + r];
1591                                }
1592
1593                            //  ***  process objects
1594
```

```
1595                            #pragma omp parallel sections num_threads(2)
1596                            {
1597                                #pragma omp section
1598                                {
1599                                    REC[0]->GetFreq();
1600                                    REC[1]->GetFreq();
1601                                    REC[2]->GetFreq();
1602                                }
1603
1604                                #pragma omp section
1605                                {
1606                                    REC[3]->GetFreq();
1607                                    REC[4]->GetFreq();
1608                                    REC[5]->GetFreq();
1609                                }
1610                            }
1611
1612                            REC[6]->GetFreq();
1613
1614                            //  ***  save freq. to disk
1615
1616                            sprintf_s(POM2, 200, "%s\\DB_REC\\%d.dat", POM1, brUz / brUzObj * 300 + PROLAZ + 1);
1617                            if (fopen_s(&stream, POM2, "w+b") == 0) {
1618                                for (i = 0; i < brObjects; i++)
1619                                    fwrite(REC[i]->X, sizeof(double), (brFreq), stream);
1620                                fclose(stream);
1621                            }
1622                            else {
1623                                PROLAZ = -5;
1624                                goto LAB3;
1625                            }
1626
1627                            PROLAZ++;
```

```
1628
1629                        } // end while()
1630
1631                        if (PROLAZ < 300) {
1632                            sprintf_s(POM2, 200, ".\\MEMBERS\\NEW");
1633                            rename(POM1, POM2);
1634                            system("rmdir /Q /S .\\MEMBERS\\NEW");
1635                            goto LAB3;
1636                        }
1637
1638                        //  ***  change data about databases and reboot databases with new member
1639
1640                        if (PROLAZ == 300) {
1641
1642                            brUz += brUzObj;
1643                            if (SpremiPodatke(0) == 0) {
1644                                PROLAZ = -7;
1645                                goto LAB3;
1646                            }
1647                            brUz -= brUzObj;
1648
1649                            DestroyDynamicFields();
1650                            povrat = CreateDynamicFields();
1651                            if (povrat == 0) {
1652                                PROLAZ = -8;
1653                                goto LAB3;
1654                            }
1655                            PROLAZ = -9;
1656                            pprolaz = 1;
1657
1658                            InvalidateRect(hWnd, NULL, TRUE);
1659                        }
1660
```

```
1661                    }  // end if()
1662
1663  LAB3:             KAMERA = 1;
1664
1665                }  //  pragma inner
1666            }  //  pragma outer
1667
1668            if (PROLAZ < 0) {
1669
1670                switch (PROLAZ) {
1671
1672                case (-1):
1673                    MessageBox(hWnd, TEXT("Memory allocation for image failed !"), TEXT("FaceRec9 -> Message:"),
                        MB_OK);
1674                    break;
1675
1676                case (-2):
1677                    MessageBox(hWnd, TEXT("Image not saved !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1678                    break;
1679
1680                case (-3):
1681                    MessageBox(hWnd, TEXT("Some inputs are wrong !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1682                    break;
1683
1684                case (-4):
1685                    MessageBox(hWnd, TEXT("Image not read !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1686                    break;
1687
1688                case (-5):
1689                    MessageBox(hWnd, TEXT("REC data file not saved !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1690                    break;
1691
1692                case (-6):
```

```
1693                    MessageBox(hWnd, TEXT("DET data file not saved !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1694                    break;
1695
1696                case (-7):
1697                    MessageBox(hWnd, TEXT("brUzoraka.txt or names.txt not saved !"), TEXT("FaceRec9 -> Message:"),
                           MB_OK);
1698                    break;
1699
1700                case (-8):
1701                    MessageBox(hWnd, TEXT("Dynamic arrays are not properly created !"), TEXT("FaceRec9 ->
                          Message:"), MB_OK);
1702                    break;
1703
1704                case (-9):
1705                    MessageBox(hWnd, TEXT("New DB Member succesfuly added !"), TEXT("FaceRec9 -> Message:"),
                          MB_OK);
1706                    break;
1707
1708                }
1709
1710            }
1711            else MessageBox(hWnd, TEXT("New DB Member creation interrupted !"), TEXT("FaceRec9 -> Message:"),
                  MB_OK);
1712
1713            delete[]bytes;
1714
1715            fftw_cleanup();
1716
1717            cap.release();
1718        }
1719
1720        break;
1721
```

```
1722          //   ********************************************************************************
1723          //   *****     SELECT AND DELETE DB MEMBER     ****************************************
1724          //   ********************************************************************************
1725
1726          case ID_RECOGNITIONDB_SELECTANDDELETEDBMEMBER:
1727          {
1728              if (KAMERA == 0) break;
1729              KAMERA = 0;
1730
1731              selClan = -1;
1732
1733              DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG4), hWnd, IzbaciClanBaze);
1734
1735              if (selClan == -1) {
1736                  KAMERA = 1;
1737                  break;
1738              }
1739
1740              i = MessageBox(hWnd, TEXT("Do You want to delete selected DB Member ?"), TEXT("FaceRec9 -> Message:"),
                     MB_OKCANCEL | MB_ICONQUESTION);
1741              if (i == IDCANCEL) {
1742                  KAMERA = 1;
1743                  break;
1744              }
1745
1746              DisableProcessWindowsGhosting();
1747
1748              j = selClan * brUzObj;
1749              for (i = j; i < j + brUzObj; i++) {
1750                  sprintf_s(POM, 200, ".\\DAT_DB_REC\\%d.dat", i + 1);
1751                  DeleteFileA(POM);
1752                  sprintf_s(POM, 200, ".\\MEMBERS\\%d.bmp", i + 1);
1753                  DeleteFileA(POM);
```

```
1754            }
1755
1756            for (i = j + brUzObj; i < brUz; i++) {
1757                sprintf_s(POM, 200, ".\\DAT_DB_REC\\%d.dat", i + 1);
1758                sprintf_s(POM1, 200, ".\\DAT_DB_REC\\%d.dat", i + 1 - brUzObj);
1759                rename(POM, POM1);
1760                sprintf_s(POM, 200, ".\\MEMBERS\\%d.bmp", i + 1);
1761                sprintf_s(POM1, 200, ".\\MEMBERS\\%d.bmp", i + 1 - brUzObj);
1762                rename(POM, POM1);
1763            }
1764
1765            for (k = selClan + 1; k < brUz / brUzObj; k++) {
1766                strcpy_s(POM, names[k]);
1767
1768                i = 0;
1769                while (POM[i] != ' ') i++;
1770                for (j = ++i; j < strlen(POM); j++)
1771                    POM[j - i] = POM[j];
1772                POM[j - i] = '\0';
1773
1774                for (j = 0; j < 300; j++) {
1775
1776                    sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\DB_REC\\%d.dat", POM, k * 300 + j + 1);
1777                    sprintf_s(POM2, 200, ".\\MEMBERS\\%s\\DB_REC\\%d.dat", POM, k * 300 + j - 300 + 1);
1778                    rename(POM1, POM2);
1779
1780                    sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\%d.bmp", POM, k * 300 + j + 1);
1781                    sprintf_s(POM2, 200, ".\\MEMBERS\\%s\\%d.bmp", POM, k * 300 + j - 300 + 1);
1782                    rename(POM1, POM2);
1783                }
1784
1785                for (j = 0; j < 30; j++) {
1786                    sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\Members30\\%d.bmp", POM, k * 30 + j + 1);
```

```
1787                    sprintf_s(POM2, 200, ".\\MEMBERS\\%s\\Members30\\%d.bmp", POM, k * 30 + j - 30 + 1);
1788                    rename(POM1, POM2);
1789                }
1790
1791            }
1792
1793            strcpy_s(POM, names[selClan]);
1794
1795            i = 0;
1796            while (POM[i] != ' ') i++;
1797            for (j = ++i; j < strlen(POM); j++)
1798                POM[j - i] = POM[j];
1799            POM[j - i] = '\0';
1800
1801            j = selClan * 300 + 1;
1802            for (i = j; i < j + 300; i++) {
1803                sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\%d.bmp", POM, i);
1804                DeleteFileA(POM1);
1805                sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\DB_REC\\%d.dat", POM, i);
1806                DeleteFileA(POM1);
1807            }
1808
1809            j = selClan * 30 + 1;
1810            for (i = j; i < j + 30; i++) {
1811                sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\Members30\\%d.bmp", POM, i);
1812                DeleteFileA(POM1);
1813            }
1814
1815            sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\DB_REC", POM);
1816            RemoveDirectoryA(POM1);
1817            sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\Members30", POM);
1818            RemoveDirectoryA(POM1);
1819            sprintf_s(POM1, 200, ".\\MEMBERS\\%s", POM);
```

```
1820            RemoveDirectoryA(POM1);
1821
1822            for (i = 0; i < brUz / brUzObj; i++) {
1823                j = 0;
1824                while (names[i][j] != ' ') j++;
1825                j++; k = 0;
1826                while (names[i][j] != '\0') {
1827                    names[i][k] = names[i][j];
1828                    j++; k++;
1829                }
1830                names[i][k] = '\0';
1831            }
1832
1833            if (fopen_s(&stream, ".\\names.txt", "w+t") == 0) {
1834                j = 0;
1835                for (i = 0; i < brUz / brUzObj; i++) {
1836                    if (i == selClan) continue;
1837                    fprintf_s(stream, "%d. %s\n", j + 1, names[i]);
1838                    j++;
1839                }
1840                fprintf_s(stream, "#");
1841                fclose(stream);
1842            }
1843            else {
1844                selClan = -1;
1845                KAMERA = 1;
1846                MessageBox(hWnd, TEXT("Can not open file names.txt,\n check DB !"), TEXT("FaceRec9 -> Message:"),
                      MB_OK);
1847                break;
1848            }
1849
1850            selClan = -1;
1851
```

```
1852            DestroyDynamicFields();
1853
1854            brUz -= brUzObj;
1855            if (fopen_s(&stream, "brUzoraka.txt", "w+t") == 0) {
1856                fprintf_s(stream, "%d,%d#", brUz, brUzObj);
1857                fclose(stream);
1858            }
1859            else {
1860                KAMERA = 1;
1861                MessageBox(hWnd, TEXT("Can not open file brUzoraka.txt,\n check DB !"), TEXT("FaceRec9 ->
                    Message:"), MB_OK);
1862                break;
1863            }
1864
1865            povrat = CreateDynamicFields();
1866
1867            if (povrat == 1)
1868                MessageBox(hWnd, TEXT("DB Member deleted !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1869
1870            pprolaz = 1;
1871
1872            KAMERA = 1;
1873
1874            InvalidateRect(hWnd, NULL, TRUE);
1875        }
1876
1877        break;
1878
1879        //   ********************************************************************************
1880        //   *****     RECOMPILE RECOGNITION AND DETECTION DATABASE (MEMORY)     **********
1881        //   ********************************************************************************
1882
1883        case ID_RECOGNITIONDB_RECOMPILERECOGNITIONDATABASE:
```

```
1884        {
1885            if (KAMERA == 0) break;
1886            KAMERA = 0;
1887
1888            i = MessageBox(hWnd, TEXT("Do You want to recompile Database ?"), TEXT("FaceRec9 -> Message:"),
                  MB_OKCANCEL | MB_ICONQUESTION);
1889            if (i == IDCANCEL) {
1890                KAMERA = 1;
1891                break;
1892            }
1893
1894            DisableProcessWindowsGhosting();
1895
1896            for (i = 0; i < brObjects; i++) {
1897                REC[i] = new GetFreq_REC();
1898                REC[i]->brUz = brUz;
1899                REC[i]->sWitch = 0;
1900            }
1901
1902            PROLAZ = 0;
1903            while (PROLAZ < brUz) {
1904
1905                Mat frame;
1906
1907                bytesX = new byte[27648];  // 96x96x3
1908
1909                //  ***  load image
1910
1911                sprintf_s(POM, 200, ".\\MEMBERS\\%d.bmp", (PROLAZ + 1));
1912                frame = imread(POM, IMREAD_COLOR);
1913                if (frame.empty()) {
1914                    PROLAZ = -2;
1915                    break;
```

```
1916                }
1917
1918                //  ***  split image to RGB components
1919
1920                size_t size = frame.total() * frame.elemSize();
1921                std::memcpy(bytesX, frame.data, size * sizeof(byte));
1922
1923                for (i = 0; i < 96; i++)
1924                    for (j = 0; j < 96; j++) {
1925                        S_B[i][j] = bytesX[i * 288 + j * 3];
1926                        S_G[i][j] = bytesX[i * 288 + j * 3 + 1];
1927                        S_R[i][j] = bytesX[i * 288 + j * 3 + 2];
1928                    }
1929
1930                delete[] bytesX;
1931
1932                //  ***  resize image to 52x52 pix
1933
1934                Reduce_Image_FFTW(96, 52);
1935
1936                k = 0;
1937                for (m = 0; m < 14; m += 13)
1938                    for (n = 0; n < 27; n += 13) {
1939                        for (p = 0; p < 26; p++)
1940                            for (r = 0; r < 26; r++) {
1941                                REC[k]->SX_R[p][r] = S_R[m + p][n + r];
1942                                REC[k]->SX_G[p][r] = S_G[m + p][n + r];
1943                                REC[k]->SX_B[p][r] = S_B[m + p][n + r];
1944                            }
1945                        k++;
1946                    }
1947
1948                m = 26; n = 13;
```

```
1949                    for (p = 0; p < 26; p++)
1950                        for (r = 0; r < 26; r++) {
1951                            REC[k]->SX_R[p][r] = S_R[m + p][n + r];
1952                            REC[k]->SX_G[p][r] = S_G[m + p][n + r];
1953                            REC[k]->SX_B[p][r] = S_B[m + p][n + r];
1954                        }
1955
1956                    //  ***  process objects
1957
1958                    #pragma omp parallel sections num_threads(2)
1959                    {
1960                        #pragma omp section
1961                        {
1962                            REC[0]->GetFreq();
1963                            REC[1]->GetFreq();
1964                            REC[2]->GetFreq();
1965                        }
1966
1967                        #pragma omp section
1968                        {
1969                            REC[3]->GetFreq();
1970                            REC[4]->GetFreq();
1971                            REC[5]->GetFreq();
1972                        }
1973                    }
1974
1975                    REC[6]->GetFreq();
1976
1977                    sprintf_s(POM2, 200, ".\\DAT_DB_REC\\%d.dat", PROLAZ + 1);
1978                    if (fopen_s(&stream, POM2, "w+b") == 0) {
1979                        for (i = 0; i < brObjects; i++)
1980                            fwrite(REC[i]->X, sizeof(double), (brFreq), stream);
1981                        fclose(stream);
```

```
1982                    }
1983                    else {
1984                        PROLAZ = -5;
1985                        break;;
1986                    }
1987
1988                    PROLAZ++;
1989
1990                } // end while()
1991
1992                KAMERA = 1;
1993
1994                if (PROLAZ == -1) MessageBox(hWnd, TEXT("Problem opening the file x.dat !"), TEXT("FaceRec9 ->
                      Message:"), MB_OK);
1995                else if (PROLAZ == brUz) MessageBox(hWnd, TEXT("Recognition and Detection Database Creation
                      finished !"),
1996                    TEXT("FaceRec9 -> Message:"), MB_OK);
1997                else if (PROLAZ == -2) MessageBox(hWnd, TEXT("Problem reading image x.jpg !"), TEXT("FaceRec9 ->
                      Message:"), MB_OK);
1998                else MessageBox(hWnd, TEXT("DataBase creation interrupted !"), TEXT("FaceRec9 -> Message:"), MB_OK);
1999
2000                fftw_cleanup();
2001            }
2002
2003            break;
2004
2005            //   ********************************************************************************
2006            //   *****     RECOGNITION DB MEMBERS OVERVIEW     ******************************
2007            //   ********************************************************************************
2008
2009            case ID_OVERVIEW_RECOGNITIONDBMEMBERSOVERVIEW:
2010            {
2011                DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG3), hWnd, PregledClanovaBaze);
```

```
2012            }
2013            break;
2014
2015            //   ********************************************************************************
2016            //   *****     RECOGNITION DB MEMBERS SELECTION     ******************************
2017            //   ********************************************************************************
2018
2019            case ID_TOOLS_DBMEMBERSSELECTION:
2020            {
2021                if (KAMERA == 0) break;
2022                KAMERA = 0;
2023
2024                selClan = -1;
2025
2026                DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG5), hWnd, Create30ElementsDB);
2027
2028                if (selClan == -1) {
2029                    KAMERA = 1;
2030                    break;
2031                }
2032
2033                i = MessageBox(hWnd, TEXT("Do You want to create 30 elements DB Member ?"), TEXT("FaceRec9 ->
                      Message:"), MB_OKCANCEL | MB_ICONQUESTION);
2034                if (i == IDCANCEL) {
2035                    KAMERA = 1;
2036                    break;
2037                }
2038
2039                DisableProcessWindowsGhosting();
2040
2041                PROLAZ = 0;
2042
2043                strcpy_s(POM, names[selClan]);
```

```
2044
2045            i = 0;
2046            while (POM[i] != ' ') i++;
2047            for (j = ++i; j < strlen(POM); j++)
2048                POM[j - i] = POM[j];
2049            POM[j - i] = '\0';
2050
2051            //  ***  create dynamic fields
2052
2053            int* put = new int[300]();
2054
2055            double** G = new double* [300];
2056            for (i = 0; i < 300; i++)
2057                G[i] = new double[300]();
2058
2059            double*** X_REC = new double** [300];
2060            for (i = 0; i < 300; i++) {
2061                X_REC[i] = new double* [(size_t)brObjects];
2062                for (j = 0; j < brObjects; j++) {
2063                    X_REC[i][j] = new double[brFreq];
2064                }
2065            }
2066
2067            //  ***  load recognition DB data
2068
2069            for (i = 0; i < 300; i++) {
2070
2071                sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\DB_REC\\%d.dat", POM, selClan * 300 + i + 1);
2072                if (fopen_s(&stream, POM1, "r+b") == 0) {
2073                    for (j = 0; j < brObjects; j++)
2074                        fread(X_REC[i][j], sizeof(double), (brFreq), stream);
2075                    fclose(stream);
2076                }
```

```
2077                else {
2078                    PROLAZ = -1;
2079                    goto LAB4;
2080                }
2081            }
2082
2083            //  ***  calculate mutual differences between elements
2084            //  ***  in the recognition database
2085            //  ***  creation of G array
2086
2087            for (i = 0; i < 300; i++)
2088                for (j = 0; j < 300; j++) {
2089                    for (m = 0; m < brObjects; m++) {
2090                        pom = 0;
2091                        for (n = 0; n < brFreq; n++)
2092                            pom += fabs(X_REC[i][m][n] - X_REC[j][m][n]);
2093                        G[i][j] += exp(-pow(pom, 10) / 2.0E-7);
2094                    }
2095
2096                    G[i][j] /= brObjects;
2097                }
2098
2099            //  ***  order elements according to distances  ***
2100
2101            Distances(G, put);
2102 /*
2103            //  ***  save order for selected member
2104
2105            sprintf_s(POM1, 200, "%s_All.txt", POM);
2106            if (fopen_s(&stream, POM1, "w+t") == 0) {
2107                for (i = 0; i < 300; i++)
2108                    fprintf_s(stream, "%d\n", put[i] + 1);
2109                fclose(stream);
```

```
2110            }
2111            else {
2112                PROLAZ = -2;
2113                goto LAB4;
2114            }
2115
2116            //  ***  pick 30 elements from order and save  ***
2117
2118            poz = 5;
2119            sprintf_s(POM1, 200, "%s.txt", POM);
2120            if (fopen_s(&stream, POM1, "w+t") == 0) {
2121                for (i = poz; i < 300; i += 10)
2122                    fprintf_s(stream, "%d\n", put[i] + 1);
2123                fclose(stream);
2124            }
2125            else {
2126                PROLAZ = -3;
2127                goto LAB4;
2128            }
2129 */
2130            //  ***  select 30 images out of 300 according to the selected elements, and renumber  ***
2131
2132            j = 0;
2133            for (i = 5; i < 300; i += 10) {
2134                sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\%d.bmp", POM, selClan * 300 + put[i] + 1);
2135                Mat frame = imread(POM1, IMREAD_COLOR);
2136                if (frame.empty()) {
2137                    PROLAZ = -4;
2138                    goto LAB4;
2139                }
2140
2141                sprintf_s(POM1, 200, ".\\MEMBERS\\%s\\MEMBERS30\\%d.bmp", POM, selClan * brUzObj + j + 1);
2142                if (!imwrite(POM1, frame)) {
```

```
2143                        PROLAZ = -5;
2144                        goto LAB4;
2145                    }
2146
2147                    sprintf_s(POM1, 200, ".\\MEMBERS\\%d.bmp", selClan * brUzObj + j + 1);
2148                    if (!imwrite(POM1, frame)) {
2149                        PROLAZ = -6;
2150                        goto LAB4;
2151                    }
2152
2153                    j++;
2154                }
2155
2156    LAB4:       switch (PROLAZ) {
2157
2158                case (0):
2159                    MessageBox(hWnd, TEXT("Member DB elements selected !"), TEXT("FaceRec9 -> Message:"), MB_OK);
2160                    break;
2161
2162                case (-1):
2163                    MessageBox(hWnd, TEXT("Recognition DB not loaded !"), TEXT("FaceRec9 -> Message:"), MB_OK);
2164                    break;
2165
2166                case (-2):
2167                    MessageBox(hWnd, TEXT("Can not open member_All.txt !"), TEXT("FaceRec9 -> Message:"), MB_OK);
2168                    break;
2169
2170                case (-3):
2171                    MessageBox(hWnd, TEXT("Can not open selected member.txt !"), TEXT("FaceRec9 -> Message:"), MB_OK);
2172                    break;
2173
2174                case (-4):
2175                    MessageBox(hWnd, TEXT("Can not read selected images !"), TEXT("FaceRec9 -> Message:"), MB_OK);
```

```
2176                    break;
2177
2178                case (-5):
2179                    MessageBox(hWnd, TEXT("Can not write Members30 selected images !"), TEXT("FaceRec9 -> Message:"),
                          MB_OK);
2180                    break;
2181
2182                case (-6):
2183                    MessageBox(hWnd, TEXT("Can not write selected images !"), TEXT("FaceRec9 -> Message:"), MB_OK);
2184                    break;
2185                }
2186
2187                //  ***  destroy dynamic arrays
2188
2189                delete[] put;
2190
2191                for (i = 0; i < 300; i++) {
2192                    for (j = 0; j < brObjects; j++)
2193                        delete[] X_REC[i][j];
2194                    delete[] X_REC[i];
2195                }
2196                delete[] X_REC;
2197
2198                for (i = 0; i < 300; i++)
2199                    delete[] G[i];
2200                delete[] G;
2201
2202                KAMERA = 1;
2203            }
2204
2205            break;
2206
2207            case ID_TOOLS_MONITORSELECT:
```

```
2208            {
2209                DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG6), hWnd, CheckBoxes);
2210
2211                if (cancel == 1) break;
2212
2213                sprintf_s(POM, 200, ".\\monitors.dat");
2214                if (fopen_s(&stream, POM, "w+b") == 0) {
2215                    fwrite(checkBoxes, sizeof(LRESULT), (2), stream);
2216                    fclose(stream);
2217                }
2218                else {
2219                    MessageBox(hWnd, TEXT("Can not open monitors.dat !"), TEXT("FaceRec9 -> Message:"), MB_OK);
2220                    break;
2221                }
2222
2223                if (checkBoxes[1] == BST_CHECKED) {
2224                    XX = GetSystemMetrics(SM_CXSCREEN);
2225                    YY = -GetSystemMetrics(SM_CYSCREEN);
2226                }
2227                else {
2228                    XX = 0;
2229                    YY = 0;
2230                }
2231
2232                if (fopen_s(&stream, "resolution.txt", "w+t") == 0) {
2233                    fprintf_s(stream, "%d   %d", XX, YY);
2234                    fclose(stream);
2235                }
2236            }
2237
2238            break;
2239
2240            case IDM_ABOUT:
```

```
2241
2242                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
2243                break;
2244
2245            case IDM_EXIT:
2246
2247                DestroyWindow(hWnd);
2248                break;
2249
2250            default:
2251                return DefWindowProc(hWnd, message, wParam, lParam);
2252            }
2253        }
2254        break;
2255
2256
2257
2258        case WM_PAINT:
2259        {
2260            PAINTSTRUCT ps;
2261            HDC hdc = BeginPaint(hWnd, &ps);
2262
2263            HGDIOBJ penGray = CreatePen(NULL, 1, RGB(0, 0, 0));
2264            SelectObject(hdc, GetStockObject(DC_BRUSH));
2265            SetDCBrushColor(hdc, RGB(95, 127, 191));
2266            SelectObject(hdc, penGray);
2267            Rectangle(hdc, 0, 0, 468, 90);
2268
2269            SelectObject(hdc, GetStockObject(OEM_FIXED_FONT));
2270            SetBkColor(hdc, RGB(95, 127, 191));
2271            SetTextColor(hdc, RGB(0, 0, 0));
2272
2273            sprintf_s(POM, 200, "total Nr. of Patterns: %d", brUz);
```

```
2274            TextOutA(hdc, 10, 20, POM, (int)strlen(POM));
2275
2276            sprintf_s(POM, 200, "Nr. of Patterns per Object: %d", brUzObj);
2277            TextOutA(hdc, 10, 40, POM, (int)strlen(POM));
2278
2279            if (povrat == 1) {
2280                SetTextColor(hdc, RGB(63, 191, 63));
2281                sprintf_s(POM, 200, "Dynamic arrays properly created !");
2282                TextOutA(hdc, 10, 60, POM, (int)strlen(POM));
2283            }
2284            else {
2285                SetTextColor(hdc, RGB(191, 63, 63));
2286                sprintf_s(POM, 200, "Dynamic arrays not created !");
2287                TextOutA(hdc, 10, 60, POM, (int)strlen(POM));
2288            }
2289
2290            HGDIOBJ penGreen = CreatePen(NULL, 1, RGB(63, 223, 63));
2291            SelectObject(hdc, GetStockObject(NULL_BRUSH));
2292            SelectObject(hdc, penGreen);
2293            Rectangle(hdc, 308, 15, 452, 79);
2294
2295            // TODO: Add any drawing code that uses hdc here...
2296            EndPaint(hWnd, &ps);
2297        }
2298        break;
2299
2300        case WM_DESTROY:
2301
2302            KAMERA = 1;
2303            waitKey(300);
2304            if (povrat == 1) {
2305                DestroyDynamicFields();
2306            }
```

```
2307
2308            PostQuitMessage(0);
2309            break;
2310
2311        default:
2312            return DefWindowProc(hWnd, message, wParam, lParam);
2313        }
2314        return 0;
2315    }
2316
2317    // Message handler for about box.
2318    INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
2319    {
2320        UNREFERENCED_PARAMETER(lParam);
2321        switch (message)
2322        {
2323        case WM_INITDIALOG:
2324            return (INT_PTR)TRUE;
2325
2326        case WM_COMMAND:
2327            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
2328            {
2329                EndDialog(hDlg, LOWORD(wParam));
2330                return (INT_PTR)TRUE;
2331            }
2332            break;
2333        }
2334        return (INT_PTR)FALSE;
2335    }
2336
2337    INT_PTR CALLBACK UnesiIme(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
2338
2339        HWND hDlg1;
```

```
2340      UNREFERENCED_PARAMETER(lParam);
2341      int nLen;
2342
2343      switch (message) {
2344      case WM_INITDIALOG:
2345          hDlg1 = GetDlgItem(hDlg, IDC_EDIT1);
2346          SetFocus(hDlg1);
2347          return (INT_PTR)FALSE;
2348
2349      case WM_COMMAND:
2350          switch (LOWORD(wParam)) {
2351          case IDOK:
2352              hDlg1 = GetDlgItem(hDlg, IDC_EDIT1);  // handle to dialog box edit field
2353              nLen = GetWindowTextLengthA(hDlg1);
2354              GetDlgItemTextA(hDlg, IDC_EDIT1, POM, nLen + 1);
2355              POM[nLen] = '\0';
2356              cancel = 0;
2357              EndDialog(hDlg, LOWORD(wParam));
2358              break;
2359
2360          case IDCANCEL:
2361              cancel = 1;
2362              EndDialog(hDlg, LOWORD(wParam));
2363              break;
2364          }
2365          break;
2366
2367      default:
2368          return (INT_PTR)FALSE;
2369      }
2370
2371      return (INT_PTR)TRUE;
2372  }
```

```
2373
2374 INT_PTR CALLBACK PregledClanovaBaze(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
2375
2376     int selClan1;
2377
2378     switch (message)
2379     {
2380     case WM_INITDIALOG:
2381     {
2382         // Add items to list.
2383         HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
2384         for (int i = 0; i < brUz / brUzObj; i++)
2385         {
2386             int pos = (int)SendMessageA(hwndList, LB_ADDSTRING, 0, (LPARAM)names[i]);
2387             SendMessage(hwndList, LB_SETITEMDATA, pos, (LPARAM)i);
2388         }
2389
2390         selClan1 = -1;
2391
2392         // Set input focus to the list box.
2393         SetFocus(hwndList);
2394         return TRUE;
2395     }
2396
2397     case WM_COMMAND:
2398
2399         switch (LOWORD(wParam))
2400         {
2401
2402         case IDOK:
2403
2404             EndDialog(hDlg, LOWORD(wParam));
2405             return TRUE;
```

```
2406
2407            case IDC_LIST1:
2408
2409                switch (HIWORD(wParam))
2410                {
2411                case LBN_SELCHANGE:
2412
2413                    HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
2414                    HWND hwndList1 = GetDlgItem(hDlg, IDC_STATIC);
2415
2416                    // Get selected index.
2417                    int lbItem = (int)SendMessage(hwndList, LB_GETCURSEL, 0, 0);
2418
2419                    // Get item data.
2420                    selClan1 = (int)SendMessage(hwndList, LB_GETITEMDATA, lbItem, 0);
2421
2422                    ShowMemberImage(hwndList1, selClan1);
2423                    return TRUE;
2424                }
2425
2426                return TRUE;
2427            }
2428
2429        case WM_DESTROY:
2430
2431            EndDialog(hDlg, LOWORD(wParam));
2432            return TRUE;
2433        }
2434
2435        return FALSE;
2436    }
2437
2438    INT_PTR CALLBACK IzbaciClanBaze(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
```

```
2439
2440        switch (message)
2441        {
2442        case WM_INITDIALOG:
2443        {
2444            // Add items to list.
2445            HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
2446            for (int i = 0; i < brUz / brUzObj; i++)
2447            {
2448                int pos = (int)SendMessageA(hwndList, LB_ADDSTRING, 0, (LPARAM)names[i]);
2449                SendMessage(hwndList, LB_SETITEMDATA, pos, (LPARAM)i);
2450            }
2451
2452            selClan = -1;
2453            // Set input focus to the list box.
2454            SetFocus(hwndList);
2455            return TRUE;
2456        }
2457
2458        case WM_COMMAND:
2459
2460            switch (LOWORD(wParam))
2461            {
2462
2463            case IDCANCEL:
2464                selClan = -1;
2465            case IDOK:
2466
2467                EndDialog(hDlg, LOWORD(wParam));
2468                return TRUE;
2469
2470            case IDC_LIST1:
2471
```

```
2472                switch (HIWORD(wParam))
2473                {
2474                case LBN_SELCHANGE:
2475
2476                    HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
2477                    HWND hwndList1 = GetDlgItem(hDlg, IDC_STATIC);
2478
2479                    // Get selected index.
2480                    int lbItem = (int)SendMessage(hwndList, LB_GETCURSEL, 0, 0);
2481
2482                    // Get item data.
2483                    selClan = (int)SendMessage(hwndList, LB_GETITEMDATA, lbItem, 0);
2484
2485                    ShowMemberImage(hwndList1, selClan);
2486                    return TRUE;
2487                }
2488
2489                return TRUE;
2490            }
2491        }
2492
2493        return FALSE;
2494    }
2495
2496    INT_PTR CALLBACK Create30ElementsDB(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
2497
2498        switch (message)
2499        {
2500        case WM_INITDIALOG:
2501        {
2502            // Add items to list.
2503            HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
2504            for (int i = 0; i < brUz / brUzObj; i++)
```

```
2505            {
2506                int pos = (int)SendMessageA(hwndList, LB_ADDSTRING, 0, (LPARAM)names[i]);
2507                SendMessage(hwndList, LB_SETITEMDATA, pos, (LPARAM)i);
2508            }
2509
2510            selClan = -1;
2511            // Set input focus to the list box.
2512            SetFocus(hwndList);
2513            return TRUE;
2514        }
2515
2516        case WM_COMMAND:
2517
2518            switch (LOWORD(wParam))
2519            {
2520
2521            case IDCANCEL:
2522                selClan = -1;
2523            case IDOK:
2524                //          destroyWindow("MemberImage");
2525                EndDialog(hDlg, LOWORD(wParam));
2526                return TRUE;
2527
2528            case IDC_LIST1:
2529
2530                switch (HIWORD(wParam))
2531                {
2532                case LBN_SELCHANGE:
2533
2534                    HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
2535                    HWND hwndList1 = GetDlgItem(hDlg, IDC_STATIC);
2536
2537                    // Get selected index.
```

```
2538                    int lbItem = (int)SendMessage(hwndList, LB_GETCURSEL, 0, 0);
2539
2540                    // Get item data.
2541                    selClan = (int)SendMessage(hwndList, LB_GETITEMDATA, lbItem, 0);
2542
2543                    ShowMemberImage(hwndList1, selClan);  //  hwndList1
2544                    return TRUE;
2545                }
2546
2547                return TRUE;
2548            }
2549        }
2550
2551        return FALSE;
2552    }
2553
2554    INT_PTR CALLBACK CheckBoxes(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
2555
2556        HWND hDlg1;
2557        UNREFERENCED_PARAMETER(lParam);
2558        int i;
2559
2560        switch (message) {
2561        case WM_INITDIALOG:
2562            for (i = 0; i < 2; i++) {
2563                hDlg1 = GetDlgItem(hDlg, i + IDC_RADIO1);
2564                SendMessage(hDlg1, BM_SETCHECK, checkBoxes[i], 0);
2565            }
2566            return (INT_PTR)TRUE;
2567
2568        case WM_COMMAND:
2569            switch (LOWORD(wParam)) {
2570
```

```
2571            case IDOK:
2572                for (i = 0; i < 2; i++) {
2573                    hDlg1 = GetDlgItem(hDlg, i + IDC_RADIO1);
2574                    checkBoxes[i] = SendMessage(hDlg1, BM_GETCHECK, 0, 0);
2575                }
2576                EndDialog(hDlg, LOWORD(wParam));
2577                break;
2578
2579            case IDCANCEL:
2580                cancel = 1;
2581                EndDialog(hDlg, LOWORD(wParam));
2582                break;
2583            }
2584            break;
2585        default:
2586            return (INT_PTR)FALSE;
2587        }
2588        return (INT_PTR)TRUE;
2589    }
2590
```