```cpp
// VoiceRecFFTW5.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "math.h"
#include "fftw3.h"
#include "list"
#include "VoiceRecFFTW5.h"

constexpr auto MAX_LOADSTRING = 100;

int M = 24000;  //  time domain max signal size
int MD = 12000;  // reduced (filtered) time domain max signal size
int N = 2400;  //  time domain rec. pattern size
int ND = 1200;  //  reduced (filtered) time domain rec. pattern size
int K = 600;  //  rec. time domain scanning buffer size
int KX = 900;  //  rec. DB freq. buffer size
int SW = 0;
int rez; //  selected pattern nr. in DB
int nrSample;  // counter
int nrDbPatternsTotal = 210;  //  total number of patterns in DB
int nrDbPatternsPerVoice = 30;  //  number of patterns per voice in DB
short POM[24000];  //  time domain signal buffer
char POM1[200], ch;
double X[900];  // freq. buffer
double R[210][900];  //  rec. DB array
double S[24000];  //  reduced signal buffer
double SX[1800], SX1[1800];  //  pattern buffer
double largest;  // max probab. of pattern

char voices[10][12] = { "1. A\0", "2. E\0", "3. I\0", "4. O\0", "5. U\0",
                        "6. TA(1)\0", "7. TE(2)\0", "8. TI(3)\0", "9. TO(4)\0", "10. TU(5)\0" };

```

```
34  int selClan;
35
36  fftw_plan plan;
37
38  void NormPNN() {
39
40      int i;
41      double sum;
42
43      sum = 0;
44      for (i = 0; i < KX; i++)    sum += X[i];
45      if (sum > 0)
46          for (i = 0; i < KX; i++) X[i] /= sum;
47  }
48
49  void PNN() {
50
51      int i, j;
52      double sum;
53
54      rez = -1; largest = 0;
55      for (i = 0; i < nrDbPatternsTotal; i++) {
56          sum = 0;
57          for (j = 0; j < KX; j++) {
58              sum += fabs(X[j] - R[i][j]);
59          }
60
61          sum = exp(-pow(sum, 8) / 0.0002);
62
63          if (sum > largest) {
64              largest = sum;
65              rez = i;
66          }
```

```
67      }
68  }
69
70  void ProcessResult() {
71
72      int rez1;
73
74      rez1 = rez / nrDbPatternsPerVoice + 1;
75
76      switch (rez1) {
77
78      case 0:
79
80          ch = '$';
81          break;
82
83      case 1:
84
85          ch = 'a';
86          break;
87
88      case 2:
89
90          ch = 'e';
91          break;
92
93      case 3:
94
95          ch = 'i';
96          break;
97
98      case 4:
99
```

```
100         ch = 'o';
101         break;
102
103     case 5:
104
105         ch = 'u';
106         break;
107
108     case 6:
109
110         ch = '1';  //  "TA"
111         break;
112
113     case 7:
114
115         ch = '2';  //  "TE"
116         break;
117
118     case 8:
119
120         ch = '3';  //  "TI"
121         break;
122
123     case 9:
124
125         ch = '4';  //  "TO"
126         break;
127
128     case 10:
129
130         ch = '5';  //  "TU"
131         break;
132     }
```

```
133 }
134
135 void FFT(double* x, fftw_complex* y) {
136
137     int i;
138
139     fftw_execute(plan);
140
141     for (i = 0; i < 300; i++) {
142
143         y[i][0] = fabs((y[i][0]) / 600);
144     }
145 }
146
147 void Distances(double** G, int* put) {
148
149     std::list<int> N, path;
150     std::list<int>::iterator n, m, it, it1;
151     int i;
152
153     for (i = 0; i < 300; ++i)
154         N.push_back(i);
155
156     n = N.begin();
157     path.splice(path.end(), N, n);
158
159
160     while (!N.empty()) {
161
162         it1 = N.begin(); it = path.begin();
163         for (m = path.begin(); m != path.end(); ++m) {
164             for (n = N.begin(); n != N.end(); ++n) {
165                 if (G[*n][*m] > G[*it1][*it]) {
```

```
166                     it = m; it1 = n;
167                 }
168             }
169         }
170
171         path.splice(path.end(), N, it1);
172     }
173
174     i = 0;
175     for (n = path.begin(); n != path.end(); ++n)
176         put[i++] = *n;
177 }
178
179 void ReduceSignal(int nrSamp) {
180
181     int i;
182
183     double* x = (double*)fftw_malloc(sizeof(double) * nrSamp);
184     for (i = 0; i < nrSamp; i++)
185         x[i] = S[i];
186     fftw_complex* y = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) * ((size_t)(nrSamp / 2) + 1));
187     fftw_plan plan1 = fftw_plan_dft_r2c_1d(nrSamp, x, y, FFTW_ESTIMATE);
188
189     fftw_execute(plan1);
190
191     fftw_plan plan2 = fftw_plan_dft_c2r_1d(nrSamp / 2, y, x, FFTW_ESTIMATE);
192
193     fftw_execute(plan2);
194
195     for (i = 0; i < nrSamp / 2; i++)
196         S[i] = x[i] / nrSamp;
197
198     fftw_free(x); fftw_free(y);
```

```
199     fftw_destroy_plan(plan1); fftw_destroy_plan(plan2);
200 }
201
202 // Global Variables:
203 HINSTANCE hInst;                                // current instance
204 WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
205 WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
206
207 // Forward declarations of functions included in this code module:
208 ATOM                MyRegisterClass(HINSTANCE hInstance);
209 BOOL                InitInstance(HINSTANCE, int);
210 LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
211 INT_PTR CALLBACK Create30ElementsDB(HWND, UINT, WPARAM, LPARAM);
212 INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
213
214 int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
215     _In_opt_ HINSTANCE hPrevInstance,
216     _In_ LPWSTR    lpCmdLine,
217     _In_ int       nCmdShow)
218 {
219     UNREFERENCED_PARAMETER(hPrevInstance);
220     UNREFERENCED_PARAMETER(lpCmdLine);
221
222     // TODO: Place code here.
223
224     // Initialize global strings
225     LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
226     LoadStringW(hInstance, IDC_VOICERECFFTW5, szWindowClass, MAX_LOADSTRING);
227     MyRegisterClass(hInstance);
228
229     // Perform application initialization:
230     if (!InitInstance(hInstance, nCmdShow))
231     {
```

```
232         return FALSE;
233     }
234
235     HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_VOICERECFFTW5));
236
237     MSG msg;
238
239     // Main message loop:
240     while (GetMessage(&msg, nullptr, 0, 0))
241     {
242         if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
243         {
244             TranslateMessage(&msg);
245             DispatchMessage(&msg);
246         }
247     }
248
249     return (int)msg.wParam;
250 }
251
252 ATOM MyRegisterClass(HINSTANCE hInstance)
253 {
254     WNDCLASSEXW wcex{};
255
256     wcex.cbSize = sizeof(WNDCLASSEX);
257
258     wcex.style = CS_HREDRAW | CS_VREDRAW;
259     wcex.lpfnWndProc = WndProc;
260     wcex.cbClsExtra = 0;
261     wcex.cbWndExtra = 0;
262     wcex.hInstance = hInstance;
263     wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_VOICERECFFTW5));
264     wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
```

```
265     wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
266     wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_VOICERECFFTW5);
267     wcex.lpszClassName = szWindowClass;
268     wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
269
270     return RegisterClassExW(&wcex);
271 }
272
273 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
274 {
275     hInst = hInstance; // Store instance handle in our global variable
276
277     HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
278         CW_USEDEFAULT, 0, 900 + 16, 384 + 59, nullptr, nullptr, hInstance, nullptr);
279
280     if (!hWnd)
281     {
282         return FALSE;
283     }
284
285     ShowWindow(hWnd, nCmdShow);
286     UpdateWindow(hWnd);
287
288     return TRUE;
289 }
290
291 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
292 {
293     static int cxClient;
294     static int cyClient;
295     int i, j, m;
296     FILE* stream;
297
```

```
298     switch (message)
299     {
300     case WM_COMMAND:
301     {
302         int wmId = LOWORD(wParam);
303         // Parse the menu selections:
304         switch (wmId)
305         {
306         case ID_FILE_LOADDATABASE:
307         {
308             i = MessageBox(hWnd, TEXT("Do You want to load DB ?"), TEXT("VoiceRecFFTW4 -> Message:"), MB_OKCANCEL |
                   MB_ICONQUESTION);
309             if (i == IDCANCEL) {
310                 break;
311             }
312
313             DisableProcessWindowsGhosting();
314
315             //  **********   load voice recog. DB   **********
316
317             int mess = 0; SW = 1;
318             for (m = 0; m < nrDbPatternsTotal; m++) {
319                 sprintf_s(POM1, 100, ".\\DAT\\%d.dat", m + 1);
320                 if ((fopen_s(&stream, POM1, "r+b") == 0) && (stream != NULL)) {
321                     fread(R[m], sizeof(double), 900, stream);
322                     fclose(stream);
323                 }
324                 else {
325                     mess = -1;
326                     MessageBox(hWnd, TEXT("Problem opening the file x.dat !"), TEXT("VoiceRecFFTW5 -> Message:"),
                          MB_OK);
327                     break;
328                 }
```

```
329             }
330
331             //  **********   select freq. pattern to show (TA)   **********
332
333             for (i = 0; i < 900; i++) X[i] = R[46][i];
334
335             InvalidateRect(hWnd, NULL, TRUE);
336
337             if (mess == 0) MessageBox(hWnd, TEXT("DB succesfully loaded !"), TEXT("VoiceRecFFTW5 -> Message:"),
                  MB_OK);
338         }
339
340         break;
341
342         case ID_RECOGNITION_RECOGNIZE:
343         {
344             i = MessageBox(hWnd, TEXT("Do You want to start recognition ?"), TEXT("VoiceRecFFTW5 -> Message:"),
                  MB_OKCANCEL | MB_ICONQUESTION);
345             if (i == IDCANCEL) {
346                 break;
347             }
348
349             DisableProcessWindowsGhosting();
350
351             //  **********   create dynamic arrays for FFTW   **********
352
353             double* x = (double*)fftw_malloc(sizeof(double) * 600);
354             fftw_complex* y = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) * ((size_t)300 + 1));
355
356             plan = fftw_plan_dft_r2c_1d(600, x, y, FFTW_ESTIMATE);
357
358             //  **********   load signal for recognition   **********
359
```

```
360             int mess = 0;  SW = 0;
361             if ((fopen_s(&stream, "signal.pcm", "r+b") == 0) && (stream != NULL)) {
362                 nrSample = (unsigned int)fread(POM, sizeof(short), (M), stream);
363                 fclose(stream);
364             }
365             else {
366                 mess = -1;
367                 goto LAB2;
368             }
369
370             if (nrSample < N) {
371                 mess = -2;
372                 goto LAB2;
373             }
374
375             //  *****   reduce signal 2x with FFTW transform    *****
376
377             for (i = 0; i < nrSample; i++)
378                 S[i] = (double)POM[i];
379
380             ReduceSignal(nrSample);
381
382             //  **********   open result file   **********
383
384             if ((fopen_s(&stream, "result.txt", "w+t") == 0) && (stream != NULL)) {
385
386                 //  **********   save some data in the result file   **********
387
388                 fprintf_s(stream, "\n char   nr.     probab.  dbPatternNr.  \n");
389
390                 //  **********   recognition loop   **********
391
392                 m = 0;
```

```
393                 while (m < nrSample / 2 - 1200) {
394
395                     //   **********   clear rec. freq. buffer   **********
396
397                     for (i = 0; i < 900; i++) X[i] = 0;
398
399                     for (i = 0; i < 1200; i++) {
400                         SX[i] = S[m + i];
401                     }
402
403                     //  **********   scan first rec. subpattern   **********
404
405                     j = 0;
406                     for (i = 0; i < 600; i++) {
407                         SX1[i] = SX[j++];
408                     }
409                     j = 599;
410                     for (i = 600; i < 1200; i++) {
411                         SX1[i] = SX[j--];
412                     }
413                     j = 0;
414                     for (i = 1200; i < 1800; i++) {
415                         SX1[i] = SX1[j++];
416                     }
417
418                     for (i = 0; i < 1200; i++) {
419                         for (j = 0; j < 600; j++) {
420                             x[j] = SX1[i + j];
421                         }
422                         FFT(x, y);
423                         for (j = 0; j < 300; j++) X[j] += (y[j][0]);
424                     }
425
```

```
426                     //  **********   scan second rec. subpattern   **********
427
428                     j = 300;
429                     for (i = 0; i < 600; i++) {
430                         SX1[i] = SX[j++];
431                     }
432                     j = 899;
433                     for (i = 600; i < 1200; i++) {
434                         SX1[i] = SX[j--];
435                     }
436                     j = 0;
437                     for (i = 1200; i < 1800; i++) {
438                         SX1[i] = SX1[j++];
439                     }
440
441                     for (i = 0; i < 1200; i++) {
442                         for (j = 0; j < 600; j++) {
443                             x[j] = SX1[i + j];
444                         }
445                         FFT(x, y);
446                         for (j = 300; j < 600; j++) X[j] += (y[j - 300][0]);
447                     }
448
449                     //  **********   scan third rec. subpattern   **********
450
451                     j = 600;
452                     for (i = 0; i < 600; i++) {
453                         SX1[i] = SX[j++];
454                     }
455                     j = 1199;
456                     for (i = 600; i < 1200; i++) {
457                         SX1[i] = SX[j--];
458                     }
```

```
459                     j = 0;
460                     for (i = 1200; i < 1800; i++) {
461                         SX1[i] = SX1[j++];
462                     }
463
464                     for (i = 0; i < 1200; i++) {
465                         for (j = 0; j < 600; j++) {
466                             x[j] = SX1[i + j];
467                         }
468                         FFT(x, y);
469                         for (j = 600; j < 900; j++) X[j] += (y[j - 600][0]);
470                     }
471
472                     //  **********   normalize, pitch & log  **********
473
474                     double sum = 0;
475                     for (i = 0; i < 900; i++)   sum += X[i];
476                     sum /= 900;
477                     if (sum > 0)
478                         for (i = 0; i < 900; i++) X[i] /= sum;
479
480                     for (i = 0; i < 900; i++) X[i] = log(X[i] + 1);
481
482                     //  **********   normalize rec. freq. pattern   **********
483
484                     NormPNN();
485
486                     //  **********   find pattern in the rec. DB with max probability   **********
487
488                     PNN();
489
490                     //  **********   find voice (character) for that pattern   **********
491
```

```
492                     ProcessResult();
493
494                     //  **********   save rec. pattern data to result.txt file   **********
495
496                     fprintf_s(stream, "\n  %c\t%d\t%.5f\t  %d\n", ch, m*2, largest, rez + 1);
497
498                     m += 10;
499
500                 }  //  end while()
501
502                 fclose(stream);
503
504             }  // end if()
505
506             else {
507                 mess = -3;
508             }
509
510             //  **********   destroy dynamic arrays   **********
511
512 LAB2:       fftw_free(x); fftw_free(y);
513             fftw_destroy_plan(plan);
514
515             fftw_cleanup();
516
517
518             InvalidateRect(hWnd, NULL, TRUE);
519
520             switch (mess) {
521
522             case (0):
523                 MessageBox(hWnd, TEXT("Recognition succesfully finished !"), TEXT("VoiceRecFFTW5 -> Message:"),
                    MB_OK);
```

```
524                 break;
525
526             case (-1):
527                 MessageBox(hWnd, TEXT("(-1) Problem opening the file signal.pcm !"), TEXT("VoiceRecFFTW5 ->
                      Message:"), MB_OK);
528                 break;
529
530             case (-2):
531                 MessageBox(hWnd, TEXT("(-2) Signal too small !"), TEXT("VoiceRecFFTW5 -> Message:"), MB_OK);
532                 break;
533
534             case (-3):
535                 MessageBox(hWnd, TEXT("(-3) Problem opening the file result.txt !"), TEXT("VoiceRecFFTW5 ->
                      Message:"), MB_OK);
536                 break;
537
538             }  //  end switch()
539         }
540
541         break;
542
543         case ID_CREATION_CREATEDATABASE:
544         {
545             i = MessageBox(hWnd, TEXT("Do You want to create DB ?"), TEXT("VoiceRecFFTW5 -> Message:"), MB_OKCANCEL
                    | MB_ICONQUESTION);
546             if (i == IDCANCEL) {
547                 break;
548             }
549
550             DisableProcessWindowsGhosting();
551
552             //  **********   create dynamic arrays for FFTW   **********
553
```

```
554             double* x = (double*)fftw_malloc(sizeof(double) * 600);
555             fftw_complex* y = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) * ((size_t)300 + 1));
556
557             plan = fftw_plan_dft_r2c_1d(600, x, y, FFTW_ESTIMATE);
558
559             //  **********   DB creation loop   **********
560
561             int mess = 0; SW = 0;
562             for (m = 0; m < nrDbPatternsTotal; m++) {
563
564                 //  **********   load time domain pattern   **********
565
566                 sprintf_s(POM1, 200, ".\\Baza2400-44K100\\%d.pcm", m + 1);
567                 if ((fopen_s(&stream, POM1, "r+b") == 0) && (stream != NULL)) {
568                     nrSample = (int)fread(POM, sizeof(short), (M), stream);
569                     fclose(stream);
570                 }
571                 else {
572                     mess = -1;
573                     MessageBox(hWnd, TEXT("Problem opening the file x.pcm !"), TEXT("VoiceRecFFTW5 -> Message:"),
                        MB_OK);
574                     break;
575                 }
576
577                 if (nrSample < 2400) {
578                     mess = -2;
579                     MessageBox(hWnd, TEXT("Signal too small !"), TEXT("VoiceRecFFTW5 -> Message:"), MB_OK);
580                     break;
581                 }
582
583                 //   *****   reduce (filter) time domain rec. pattern   *****
584
585                 for (i = 0; i < 2400; i++)
```

```
586                     S[i] = (double)POM[i];
587
588                 ReduceSignal(2400);
589
590                 for (i = 0; i < 1200; i++)
591                     SX[i] = S[i];
592
593                 //  **********   clear freq. buffer   **********
594
595                 for (i = 0; i < 900; i++) {
596                     X[i] = 0;
597                 }
598
599                 //  **********   scan first rec. subpattern   **********
600
601                 j = 0;
602                 for (i = 0; i < 600; i++) {
603                     SX1[i] = SX[j++];
604                 }
605                 j = 599;
606                 for (i = 600; i < 1200; i++) {
607                     SX1[i] = SX[j--];
608                 }
609                 j = 0;
610                 for (i = 1200; i < 1800; i++) {
611                     SX1[i] = SX1[j++];
612                 }
613
614                 for (i = 0; i < 1200; i++) {
615                     for (j = 0; j < 600; j++) {
616                         x[j] = SX1[i + j];
617                     }
618                     FFT(x, y);
```

```
619                     for (j = 0; j < 300; j++) X[j] += (y[j][0]);
620                 }
621
622                 //  **********   scan second rec. subpattern   **********
623
624                 j = 300;
625                 for (i = 0; i < 600; i++) {
626                     SX1[i] = SX[j++];
627                 }
628                 j = 899;
629                 for (i = 600; i < 1200; i++) {
630                     SX1[i] = SX[j--];
631                 }
632                 j = 0;
633                 for (i = 1200; i < 1800; i++) {
634                     SX1[i] = SX1[j++];
635                 }
636
637                 for (i = 0; i < 1200; i++) {
638                     for (j = 0; j < 600; j++) {
639                         x[j] = SX1[i + j];
640                     }
641                     FFT(x, y);
642                     for (j = 0; j < 300; j++) X[j + 300] += (y[j][0]);
643                 }
644
645                 //  **********   scan third rec. subpattern   **********
646
647                 j = 600;
648                 for (i = 0; i < 600; i++) {
649                     SX1[i] = SX[j++];
650                 }
651                 j = 1199;
```

```
652                 for (i = 600; i < 1200; i++) {
653                     SX1[i] = SX[j--];
654                 }
655                 j = 0;
656                 for (i = 1200; i < 1800; i++) {
657                     SX1[i] = SX1[j++];
658                 }
659 
660                 for (i = 0; i < 1200; i++) {
661                     for (j = 0; j < 600; j++) {
662                         x[j] = SX1[i + j];
663                     }
664                     FFT(x, y);
665                     for (j = 0; j < 300; j++) X[j + 600] += (y[j][0]);
666                 }
667 
668                 //  **********   normalize, pitch & log  **********
669 
670                 double sum = 0;
671                 for (i = 0; i < 900; i++)   sum += X[i];
672                 sum /= 900;
673                 if (sum > 0)
674                     for (i = 0; i < 900; i++) X[i] /= sum;
675 
676                 for (i = 0; i < 900; i++) X[i] = log(X[i] + 1);
677 
678                 //  **********   normalize freq. pattern   **********
679 
680                 NormPNN();
681 
682                 //  **********   save normalized freq. pattern   **********
683 
684                 sprintf_s(POM1, 200, ".\\DAT\\%d.dat", m + 1);
```

```
685                 if ((fopen_s(&stream, POM1, "w+b") == 0) && (stream != NULL)) {
686                     fwrite(X, sizeof(double), 900, stream);
687                     fclose(stream);
688                 }
689                 else {
690                     mess = -3;
691                     MessageBox(hWnd, TEXT("Problem opening the file x.dat !"), TEXT("VoiceRecFFTW5 -> Message:"),
                      MB_OK);
692                     break;
693                 }
694             }
695 
696             //  **********   destroy dynamic arrays   **********
697 
698 
699             fftw_free(x); fftw_free(y);
700             fftw_destroy_plan(plan);
701 
702             fftw_cleanup();
703 
704 
705             InvalidateRect(hWnd, NULL, TRUE);
706 
707             if (mess == 0) MessageBox(hWnd, TEXT("DB succesfully created !"), TEXT("VoiceRecFFTW5 -> Message:"),
              MB_OK);
708         }
709 
710         break;
711 
712         case ID_CREATION_CREATEREDUCEDDATABASE:
713         {
714             selClan = -1;
715 
```

```
716             //  **********   select voice   **********
717
718             DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, Create30ElementsDB);
719
720             if (selClan == -1) {
721                 break;
722             }
723
724             DisableProcessWindowsGhosting();
725
726             int poz;
727
728             //  **********   create dynamic arrays   **********
729
730             int* put = new int[300]();
731
732             double** G = new double* [300];
733             for (i = 0; i < 300; i++)
734                 G[i] = new double[300]();
735
736             double** X_REC = new double* [300];
737             for (i = 0; i < 300; i++) {
738                 X_REC[i] = new double[900]();
739             }
740
741             //  **********   load recog. DB for selected voice   **********
742
743             int mess = 0;
744             for (i = 0; i < 300; i++) {
745
746                 sprintf_s(POM1, 200, ".DAT\\%d.dat", selClan * 300 + i + 1);
747                 if ((fopen_s(&stream, POM1, "r+b") == 0) && (stream != NULL)) {
748                     fread(X_REC[i], sizeof(double), 900, stream);
```

```
749                     fclose(stream);
750                 }
751                 else {
752                     mess = -1;
753                     goto LAB1;
754                 }
755             }
756 
757             //  ***  calculate mutual diferences between elements
758             //  ***  in the recognition database
759             //  ***  creation of G array
760 
761             for (i = 0; i < 300; i++)
762                 for (j = 0; j < 300; j++) {
763                     double sum = 0;
764                     for (int n = 0; n < 900; n++)
765                         sum += fabs(X_REC[i][n] - X_REC[j][n]);
766                     G[i][j] = exp(-pow(sum, 8) / 0.0002);
767                 }
768 
769             //  **********   arrange elements   **********
770 
771             Distances(G, put);
772 
773             //  **********   save 30 elements positions   **********
774 
775             poz = 5;
776             sprintf_s(POM1, 200, "%d.txt", selClan + 1);
777             if ((fopen_s(&stream, POM1, "w+t") == 0) && (stream != NULL)) {
778                 for (i = poz; i < 300; i += 10)
779                     fprintf_s(stream, "%d\n", put[i] + 1);
780                 fclose(stream);
781             }
```

```
782             else {
783                 mess = -2;
784                 goto LAB1;
785             }
786
787             j = 0;
788             for (i = poz; i < 300; i += 10) {
789
790                 //  **********   read element from recog. DB considering position   **********
791
792                 sprintf_s(POM1, 200, ".\\DAT\\%d.dat", selClan * 300 + put[i] + 1);
793                 if ((fopen_s(&stream, POM1, "r+b") == 0) && (stream != NULL)) {
794                     fread(X, sizeof(double), 900, stream);
795                     fclose(stream);
796                 }
797                 else {
798                     mess = -3;
799                     goto LAB1;
800                 }
801
802                 //  **********   save element to reduced DB dir.   **********
803
804                 sprintf_s(POM1, 200, ".\\R_DAT\\%d.dat", selClan * 30 + j + 1);
805                 if ((fopen_s(&stream, POM1, "w+b") == 0) && (stream != NULL)) {
806                     fwrite(X, sizeof(double), 900, stream);
807                     fclose(stream);
808                 }
809                 else {
810                     mess = -4;
811                     goto LAB1;
812                 }
813
814                 j++;
```

```
815             }
816
817             //  **********   destroy dynamic arrays   **********
818
819 LAB1:       delete[] put;
820
821             for (i = 0; i < 300; i++) {
822                 delete[] X_REC[i];
823             }
824             delete[] X_REC;
825
826             for (i = 0; i < 300; i++)
827                 delete[] G[i];
828             delete[] G;
829
830             switch (mess) {
831
832             case (0):
833                 MessageBox(hWnd, TEXT("Member DB elements selected !"), TEXT("VoiceRecFFTW5 -> Message:"), MB_OK);
834                 break;
835
836             case (-1):
837                 MessageBox(hWnd, TEXT("(-1) Selected member DB not loaded !"), TEXT("VoiceRecFFTW5 -> Message:"),
                      MB_OK);
838                 break;
839
840             case (-2):
841                 MessageBox(hWnd, TEXT("(-2) txt file with list of reduced DB members not saved !"), TEXT
                      ("VoiceRecFFTW5 -> Message:"), MB_OK);
842                 break;
843
844             case (-3):
845                 MessageBox(hWnd, TEXT("(-3) dat file from DAT dir not loaded !"), TEXT("VoiceRecFFTW5 ->
```

```
                    Message:"), MB_OK);
846                 break;
847
848             case (-4):
849                 MessageBox(hWnd, TEXT("(-4) dat file not saved to R_DAT dir !"), TEXT("VoiceRecFFTW5 -> Message:"),
                       MB_OK);
850                 break;
851             }
852         }
853
854         break;
855
856         case IDM_ABOUT:
857             DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
858             break;
859
860         case IDM_EXIT:
861             DestroyWindow(hWnd);
862             break;
863
864         default:
865             return DefWindowProc(hWnd, message, wParam, lParam);
866         }  //  end switch()
867     }
868     break;
869
870     case WM_SIZE:
871     {
872         cxClient = LOWORD(lParam);
873         cyClient = HIWORD(lParam);
874     }
875     break;
876
```

```
877     case WM_PAINT:
878     {
879         PAINTSTRUCT ps;
880         HDC hdc = BeginPaint(hWnd, &ps);
881 
882         MoveToEx(hdc, 0, cyClient / 2, NULL);
883         LineTo(hdc, cxClient, cyClient / 2);
884 
885         for (i = 0; i < 900; i++) {
886             MoveToEx(hdc, (int)(cxClient * ((i)) / 900), cyClient / 2, NULL);
887             LineTo(hdc, (int)(cxClient * ((i)) / 900),
888                 (int)(cyClient / 2 * (1 - (X[i] * 50))));
889         }
890 
891         if (SW == 1) {
892 
893             SelectObject(hdc, GetStockObject(OEM_FIXED_FONT));
894             SetBkColor(hdc, RGB(255, 255, 255));
895             SetTextColor(hdc, RGB(63, 63, 191));
896             sprintf_s(POM1, 200, "database Pattern TA");
897             TextOutA(hdc, 40, cyClient / 2 + 20, POM1, (int)strlen(POM1));
898         }
899 
900         // TODO: Add any drawing code that uses hdc here...
901         EndPaint(hWnd, &ps);
902     }
903     break;
904 
905     case WM_DESTROY:
906         PostQuitMessage(0);
907         break;
908 
909     default:
```

```
910         return DefWindowProc(hWnd, message, wParam, lParam);
911
912     }
913     return 0;
914 }
915
916 INT_PTR CALLBACK Create30ElementsDB(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
917
918     switch (message)
919     {
920     case WM_INITDIALOG:
921     {
922         // Add items to list.
923         HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
924         for (int i = 0; i < nrDbPatternsTotal / nrDbPatternsPerVoice; i++)
925         {
926             int pos = (int)SendMessageA(hwndList, LB_ADDSTRING, 0, (LPARAM)voices[i]);
927             SendMessage(hwndList, LB_SETITEMDATA, pos, (LPARAM)i);
928         }
929
930         selClan = -1;
931         // Set input focus to the list box.
932         SetFocus(hwndList);
933         return TRUE;
934     }
935
936     case WM_COMMAND:
937     {
938         switch (LOWORD(wParam))
939         {
940         case IDCANCEL:
941             selClan = -1;
942
```

```
943         case IDOK:
944             //          destroyWindow("MemberImage");
945             EndDialog(hDlg, LOWORD(wParam));
946             return TRUE;
947 
948         case IDC_LIST1:
949 
950             switch (HIWORD(wParam))
951             {
952             case LBN_SELCHANGE:
953 
954                 HWND hwndList = GetDlgItem(hDlg, IDC_LIST1);
955                 HWND hwndList1 = GetDlgItem(hDlg, IDC_STATIC);
956 
957                 // Get selected index.
958                 int lbItem = (int)SendMessage(hwndList, LB_GETCURSEL, 0, 0);
959 
960                 // Get item data.
961                 selClan = (int)SendMessage(hwndList, LB_GETITEMDATA, lbItem, 0);
962 
963                 return TRUE;
964             }
965         }
966         return TRUE;
967     }
968     }
969     return FALSE;
970 }
971 
972 // Message handler for about box.
973 INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
974 {
975     UNREFERENCED_PARAMETER(lParam);
```

```
976     switch (message)
977     {
978     case WM_INITDIALOG:
979         return (INT_PTR)TRUE;
980
981     case WM_COMMAND:
982         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
983         {
984             EndDialog(hDlg, LOWORD(wParam));
985             return (INT_PTR)TRUE;
986         }
987         break;
988     }
989     return (INT_PTR)FALSE;
990 }
991
```