```cpp
// CPPLinearAlgebraIM.cpp : Defines the entry point for the application.
// (c) Tuesday, January 21, 2025, by James Pate Williams, Jr.
// Reference: "Elementary Numerical Analysis an Algorithmic Approach"
// Third Edition (c) 1980 by S. D. Conte and Carl de Boor
// Chapter 5 pages 230 - 231 Gauss-Seidel and Successive Overrelaxation
// https://www3.nd.edu/~zxu2/acms40390F12/Lec-7.3.pdf

#include "framework.h"
#include "CPPLinearAlgebraIM.h"
#include <chrono>
#include <cwchar>
#include <vector>

#define MAX_LOADSTRING 100

// resource definitions

#define IDC_EDIT1 201
#define IDC_EDIT2 202
#define IDC_EDIT3 203
#define IDC_EDIT4 204
#define IDC_EDIT5 205

#define IDC_STATIC1 301
#define IDC_STATIC2 302
#define IDC_STATIC3 303
#define IDC_STATIC4 304

#define IDC_BUTTON_COMPUTE 401
#define IDC_BUTTON_CLEAR 402

// Global Variables:

HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
WCHAR buffer[16384] = { }, line[128] = { };
std::vector<double> b;              // right-hand sides
std::vector<double> x0;             // solution iterate
std::vector<double> x1;             // solution vector
std::vector<std::vector<double>> A; // example matrix

// exercise Exercises 5.3-7 page 234 of reference

double omega1 = 1.6, omega2 = 1.2;
double a1[4][4] = {
    { 4.0, -1.0, 0.0, 0.0 }, { -1.0, 4.0, -1.0, 0.0 },
    { 0.0, -1.0, 4.0, -1.0 }, { 0.0, 0.0, -1.0, 4.0 } };
double a2[3][3] = {
    { 5.0, -2.0, 3.0}, { -3.0, 9.0, 1.0 },
    { 2.0, -1.0, -7.0 } };
double b1[4] = { 1.0, 1.0, 1.0, 1.0 };
```

```
53  double b2[3] = { -1.0, 2.0, 3.0 };
54
55  // Forward declarations of functions included in this code module:
56
57  ATOM                MyRegisterClass(HINSTANCE hInstance);
58  BOOL                InitInstance(HINSTANCE, int);
59  LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
60  INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
61
62  int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
63                        _In_opt_ HINSTANCE hPrevInstance,
64                        _In_ LPWSTR    lpCmdLine,
65                        _In_ int       nCmdShow)
66  {
67      UNREFERENCED_PARAMETER(hPrevInstance);
68      UNREFERENCED_PARAMETER(lpCmdLine);
69
70      // TODO: Place code here.
71
72      // Initialize global strings
73      LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
74      LoadStringW(hInstance, IDC_CPPLINEARALGEBRAIM, szWindowClass,
          MAX_LOADSTRING);
75      MyRegisterClass(hInstance);
76
77      // Perform application initialization:
78      if (!InitInstance (hInstance, nCmdShow))
79      {
80          return FALSE;
81      }
82
83      HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE
          (IDC_CPPLINEARALGEBRAIM));
84
85      MSG msg;
86
87      // Main message loop:
88      while (GetMessage(&msg, nullptr, 0, 0))
89      {
90          if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
91          {
92              TranslateMessage(&msg);
93              DispatchMessage(&msg);
94          }
95      }
96
97      return (int) msg.wParam;
98  }
99  //
100 //  FUNCTION: MyRegisterClass()
101 //
102 //  PURPOSE: Registers the window class.
```

```
103 //
104 ATOM MyRegisterClass(HINSTANCE hInstance)
105 {
106     WNDCLASSEXW wcex = { };
107
108     wcex.cbSize = sizeof(WNDCLASSEX);
109
110     wcex.style          = CS_HREDRAW | CS_VREDRAW;
111     wcex.lpfnWndProc    = WndProc;
112     wcex.cbClsExtra     = 0;
113     wcex.cbWndExtra     = 0;
114     wcex.hInstance      = hInstance;
115     wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE
        (IDI_CPPLINEARALGEBRAIM));
116     wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
117     wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
118     wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_CPPLINEARALGEBRAIM);
119     wcex.lpszClassName  = szWindowClass;
120     wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
121
122     return RegisterClassExW(&wcex);
123 }
124 //
125 //   FUNCTION: InitInstance(HINSTANCE, int)
126 //
127 //   PURPOSE: Saves instance handle and creates main window
128 //
129 //   COMMENTS:
130 //
131 //        In this function, we save the instance handle in a global variable and
132 //        create and display the main program window.
133 //
134 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
135 {
136    hInst = hInstance; // Store instance handle in our global variable
137
138    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
139       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
140
141    if (!hWnd)
142    {
143       return FALSE;
144    }
145
146    ShowWindow(hWnd, nCmdShow);
147    UpdateWindow(hWnd);
148
149    return TRUE;
150 }
151
152 void GaussSeidel(
153     double tolerance, double dummy,
```

```
154     int maxIts, int n,
155     double& norm2x, int& its,
156     std::vector<double>& b,
157     std::vector<double>& x0,
158     std::vector<double>& x1,
159     std::vector<std::vector<double>>& A)
160 {
161     int i = 0, j = 0;
162     std::vector<double> c;              // right-hand sides
163     std::vector<std::vector<double>> B; // working storage
164 
165     c.resize(n);
166     B.resize(n);
167 
168     for (auto& row : B)
169     {
170         row.resize(n);
171     }
172 
173     for (i = 0; i < n; i++)
174     {
175         x1[i] = x0[i];
176     }
177 
178     for (i = 0; i < n; i++)
179     {
180         for (j = 0; j < n; j++)
181         {
182             if (i != j)
183             {
184                 B[i][j] = -A[i][j] / A[i][i];
185             }
186 
187             else
188             {
189                 B[i][j] = 0.0;
190             }
191         }
192 
193         c[i] = b[i] / A[i][i];
194     }
195 
196     its = 0;
197 
198     while (its < maxIts)
199     {
200         for (i = 0; i < n; i++)
201         {
202             double sum1 = 0;
203 
204             if ((i - 1) >= 0)
205             {
```

```
206                 for (j = 0; j <= i - 1; j++)
207                 {
208                     sum1 += B[i][j] * x1[j];
209                 }
210             }
211 
212             double sum2 = 0;
213 
214             if ((i + 1) < n)
215             {
216                 for (j = i + 1; j < n; j++)
217                 {
218                     sum2 += B[i][j] * x0[j];
219                 }
220             }
221 
222             x1[i] = sum1 + sum2 + c[i];
223         }
224 
225         norm2x = 0.0;
226 
227         for (i = 0; i < n; i++)
228         {
229             double delta = x1[i] - x0[i];
230 
231             norm2x += delta * delta;
232         }
233 
234         norm2x = sqrt(norm2x);
235 
236         its++;
237 
238         if (norm2x < tolerance)
239         {
240             break;
241         }
242 
243         for (i = 0; i < n; i++)
244         {
245             x0[i] = x1[i];
246         }
247     }
248 }
249 
250 void SuccessOverrelaxation(
251     double tolerance, double omega,
252     int maxIts, int n,
253     double& norm2x, int& its,
254     std::vector<double>& b,
255     std::vector<double>& x0,
256     std::vector<double>& x1,
257     std::vector<std::vector<double>>& A)
```

```
258 {
259     int i = 0, j = 0;
260 
261     for (i = 0; i < n; i++)
262     {
263         x1[i] = x0[i];
264     }
265 
266     its = 0;
267 
268     while (its < maxIts)
269     {
270         for (i = 0; i < n; i++)
271         {
272             double sum1 = 0;
273 
274             if ((i - 1) >= 0)
275             {
276                 for (j = 0; j < i; j++)
277                 {
278                     sum1 += A[i][j] * x1[j];
279                 }
280             }
281 
282             double sum2 = 0;
283 
284             if ((i + 1) < n)
285             {
286                 for (j = i + 1; j < n; j++)
287                 {
288                     sum2 += A[i][j] * x0[j];
289                 }
290             }
291 
292             x1[i] = (1.0 - omega) * x0[i] +
293                 omega * (b[i] - sum1 - sum2) / A[i][i];
294         }
295 
296         norm2x = 0.0;
297 
298         for (i = 0; i < n; i++)
299         {
300             double delta = x1[i] - x0[i];
301 
302             norm2x += delta * delta;
303         }
304 
305         norm2x = sqrt(norm2x);
306 
307         its++;
308 
309         if (norm2x < tolerance)
```

```
310         {
311             break;
312         }
313 
314         for (i = 0; i < n; i++)
315         {
316             x0[i] = x1[i];
317         }
318     }
319 }
320 
321 void Jacobi(
322     double tolerance, double dummy,
323     int maxIts, int n,
324     double& norm2x, int& its,
325     std::vector<double>& b,
326     std::vector<double>& x0,
327     std::vector<double>& x1,
328     std::vector<std::vector<double>>& A)
329 {
330     its = 0;
331 
332     while (its < maxIts)
333     {
334         for (int i = 0; i < n; i++)
335         {
336             double sum = 0.0;
337 
338             for (int j = 0; j < n; j++)
339             {
340                 if (j != i)
341                 {
342                     sum += A[i][j] * x0[j];
343                 }
344             }
345 
346             x1[i] = (b[i] - sum) / A[i][i];
347         }
348 
349         norm2x = 0.0;
350 
351         for (int i = 0; i < n; i++)
352         {
353             double delta = x1[i] - x0[i];
354 
355             norm2x += delta * delta;
356         }
357 
358         norm2x = sqrt(norm2x);
359 
360         its++;
361 
```

```
362         if (norm2x < tolerance)
363         {
364             break;
365         }
366
367         for (int i = 0; i < n; i++)
368         {
369             x0[i] = x1[i];
370         }
371     }
372 }
373
374 void Test(
375     double tolerance, double dummy,
376     int maxIts, int n, int problem,
377     double& norm2b,
378     double& norm2x, int& its,
379     std::vector<double>& b,
380     std::vector<double>& x0,
381     std::vector<double>& x1,
382     std::vector<std::vector<double>>& A,
383     void (*function)(double, double,
384         int, int, double&, int&,
385         std::vector<double>&,
386         std::vector<double>&,
387         std::vector<double>&,
388         std::vector<std::vector<double>>&))
389 {
390     int i, j;
391
392     for (i = 0; i < n; i++)
393     {
394         if (n == 4 && problem == 1 || problem == 3 || problem == 5)
395         {
396             b[i] = b1[i];
397         }
398
399         else if (n == 3 && problem == 2 || problem == 4 || problem == 6)
400         {
401             b[i] = b2[i];
402         }
403
404         x0[i] = 0.0;
405
406         for (j = 0; j < n; j++)
407         {
408             if (n == 4 && problem == 1 || problem == 3 || problem == 5)
409             {
410                 A[i][j] = a1[i][j];
411             }
412
413             else if (n == 3 && problem == 2 || problem == 4 || problem == 6)
```

```
414             {
415                 A[i][j] = a2[i][j];
416             }
417         }
418     }
419
420     function(tolerance, dummy, maxIts, n,
421         norm2x, its, b, x0, x1, A);
422
423     norm2b = 0.0;
424
425     for (i = 0; i < n; i++)
426     {
427         double sum = 0.0;
428
429         for (j = 0; j < n; j++)
430         {
431             sum += A[i][j] * x1[j];
432         }
433
434         norm2b += pow(b[i] - sum, 2.0);
435     }
436
437     norm2b = sqrt(norm2b);
438 }
439
440 //
441 //  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
442 //
443 //  PURPOSE: Processes messages for the main window.
444 //
445 //  WM_COMMAND  - process the application menu
446 //  WM_PAINT    - Paint the main window
447 //  WM_DESTROY  - post a quit message and return
448 //
449 //
450 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
451 {
452     static double norm2b = 0, norm2x = 0, tolerance = 0;
453     static int its = 0, maxIts = 0, problem;
454     static size_t i = 0, j = 0, n;
455     static WCHAR* endPt;
456     static HWND hwndEdit1 = 0, hwndEdit2 = 0;
457     static HWND hwndEdit3 = 0, hwndEdit4 = 0, hwndEdit5 = 0;
458
459     switch (message)
460     {
461     case WM_CREATE:
462     {
463         CreateWindowEx(0, L"STATIC", L"n (3 or 4):", WS_CHILD | WS_VISIBLE,
464             10, 10, 80, 20, hWnd, (HMENU)IDC_STATIC1, GetModuleHandle(NULL),
                NULL);
```

```
465         CreateWindowEx(0, L"STATIC", L"Max Its:", WS_CHILD | WS_VISIBLE,
466             10, 40, 80, 20, hWnd, (HMENU)IDC_STATIC2, GetModuleHandle(NULL),
                NULL);
467         CreateWindowEx(0, L"STATIC", L"Tolerance:", WS_CHILD | WS_VISIBLE,
468             10, 70, 80, 20, hWnd, (HMENU)IDC_STATIC3, GetModuleHandle(NULL),
                NULL);
469         CreateWindowEx(0, L"STATIC", L"Prob (1 - 4):", WS_CHILD | WS_VISIBLE,
470             10, 100, 80, 20, hWnd, (HMENU)IDC_STATIC4, GetModuleHandle(NULL),
                NULL);
471
472         hwndEdit1 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
473             100, 10, 200, 20, hWnd, (HMENU)IDC_EDIT1, GetModuleHandle(NULL),
                NULL);
474         hwndEdit2 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
475             100, 40, 200, 20, hWnd, (HMENU)IDC_EDIT2, GetModuleHandle(NULL),
                NULL);
476         hwndEdit3 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
477             100, 70, 200, 20, hWnd, (HMENU)IDC_EDIT3, GetModuleHandle(NULL),
                NULL);
478         hwndEdit4 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER,
479             100, 100, 200, 20, hWnd, (HMENU)IDC_EDIT4, GetModuleHandle(NULL),
                NULL);
480         hwndEdit5 = CreateWindowEx(0, L"EDIT", NULL, WS_CHILD | WS_VISIBLE |
              WS_BORDER
481             | ES_MULTILINE, 100, 170, 400, 400, hWnd, (HMENU)IDC_EDIT5,
482             GetModuleHandle(NULL), NULL);
483
484         CreateWindowEx(0, L"BUTTON", L"Compute", WS_CHILD | WS_VISIBLE |
              BS_PUSHBUTTON,
485             10, 130, 80, 30, hWnd, (HMENU)IDC_BUTTON_COMPUTE, GetModuleHandle
                (NULL), NULL);
486         CreateWindowEx(0, L"BUTTON", L"Clear", WS_CHILD | WS_VISIBLE |
              BS_PUSHBUTTON,
487             220, 130, 80, 30, hWnd, (HMENU)IDC_BUTTON_CLEAR, GetModuleHandle
                (NULL), NULL);
488
489         SetDlgItemText(hWnd, IDC_EDIT1, L"4");
490         SetDlgItemText(hWnd, IDC_EDIT2, L"100");
491         SetDlgItemText(hWnd, IDC_EDIT3, L"1.0e-12");
492         SetDlgItemText(hWnd, IDC_EDIT4, L"1");
493     }
494     break;
495     case WM_COMMAND:
496         {
497             int wmId = LOWORD(wParam);
498             // Parse the menu selections:
499             switch (wmId)
500             {
```

```
501             case IDC_BUTTON_COMPUTE:
502             {
503                 n = GetDlgItemInt(hWnd, IDC_EDIT1, FALSE, FALSE);
504
505                 if (n != 3 && n != 4)
506                 {
507                     MessageBox(hWnd, L"n must be 3 or 4", L"Warning",
508                         MB_OK | MB_ICONWARNING);
509                     return 0;
510                 }
511
512                 maxIts = GetDlgItemInt(hWnd, IDC_EDIT2, FALSE, FALSE);
513                 problem = GetDlgItemInt(hWnd, IDC_EDIT4, FALSE, FALSE);
514
515                 if (problem < 1 || problem > 6)
516                 {
517                     MessageBox(hWnd, L"Problem must be 1 to 6", L"Warning",
518                         MB_OK | MB_ICONWARNING);
519                     return 0;
520                 }
521
522                 GetWindowText(hwndEdit3, line, 127);
523                 tolerance = wcstod(line, &endPt);
524
525                 auto start = std::chrono::high_resolution_clock::now();
526
527                 x0.resize(n);
528                 x1.resize(n);
529                 b.resize(n);
530
531                 A.resize(n);
532
533                 for (auto& row : A)
534                 {
535                     row.resize(n);
536                 }
537
538                 if (n == 4 && problem == 1)
539                 {
540                     Test(tolerance, omega1, maxIts, n, problem,
541                         norm2b, norm2x, its, b, x0, x1, A, GaussSeidel);
542                 }
543
544                 else if (n == 3 && problem == 2)
545                 {
546                     Test(tolerance, omega2, maxIts, n, problem,
547                         norm2b, norm2x, its, b, x0, x1, A, GaussSeidel);
548                 }
549
550                 else if (n == 4 && problem == 3)
551                 {
552                     Test(tolerance, omega1, maxIts, n, problem,
```

```
553                         norm2b, norm2x, its, b, x0, x1, A,
554                         SuccessOverrelaxation);
555                 }
556
557                 else if (n == 3 && problem == 4)
558                 {
559                     Test(tolerance, omega2, maxIts, n, problem,
560                         norm2b, norm2x, its, b, x0, x1, A,
561                         SuccessOverrelaxation);
562                 }
563
564                 else if (n == 4 && problem == 5)
565                 {
566                     Test(tolerance, omega1, maxIts, n, problem,
567                         norm2b, norm2x, its, b, x0, x1, A, Jacobi);
568                 }
569
570                 else if (n == 3 && problem == 6)
571                 {
572                     Test(tolerance, omega2, maxIts, n, problem,
573                         norm2b, norm2x, its, b, x0, x1, A, Jacobi);
574                 }
575
576                 else if (n == 4)
577                 {
578                     MessageBox(hWnd, L"If n = 4 Problem must be 1 or 3 or 5",
579                         L"Warning", MB_OK | MB_ICONWARNING);
580                     return 0;
581                 }
582
583                 else if (n == 3)
584                 {
585                     MessageBox(hWnd, L"If n = 3 Problem must be 2 or 4 or 6",
586                         L"Warning", MB_OK | MB_ICONWARNING);
587                     return 0;
588                 }
589
590                 auto endTm = std::chrono::high_resolution_clock::now();
591                 auto duration = std::chrono::duration_cast<
592                     std::chrono::microseconds>(endTm - start);
593                 long long runtime = duration.count();
594
595                 if (problem == 1 || problem == 2)
596                 {
597                     wsprintf(line, L"Gauss-Seidel\r\n\r\n");
598                     wcscat_s(buffer, 16383, line);
599                 }
600
601                 else if (problem == 3 || problem == 4)
602                 {
603                     wsprintf(line, L"Successive Overrelaxation (SOR) Method\r\n\r ↵
                      \n");
```

```
604                     wcscat_s(buffer, 16383, line);
605                 }
606
607                 else if (problem == 5 || problem == 6)
608                 {
609                     wsprintf(line, L"Jacobi Method\r\n\r\n");
610                     wcscat_s(buffer, 16383, line);
611                 }
612
613                 wsprintf(line, L"Solution Vector x\r\n\r\n");
614                 wcscat_s(buffer, 16383, line);
615
616                 for (size_t i = 0; i < n; i++)
617                 {
618                     swprintf_s(line, L"%13.10lf\t", x1[i]);
619                     wcscat_s(buffer, 16383, line);
620                 }
621
622                 wsprintf(line, L"\r\n");
623                 wcscat_s(buffer, 16383, line);
624
625                 wsprintf(line, L"\r\nX - Norm - 2 = ");
626                 wcscat_s(buffer, 16383, line);
627
628                 swprintf_s(line, L"%13.10lf\r\n", norm2x);
629                 wcscat_s(buffer, 16383, line);
630
631                 wsprintf(line, L"\r\nb - Norm - 2 = ");
632                 wcscat_s(buffer, 16383, line);
633
634                 swprintf_s(line, L"%13.10lf\r\n\r\n", norm2b);
635                 wcscat_s(buffer, 16383, line);
636
637                 swprintf_s(line, L"Iterations = %ld\r\n\r\n", its);
638                 wcscat_s(buffer, 16383, line);
639
640                 swprintf_s(line, L"Runtime = %lld Microseconds\r\n\r\n",
641                     runtime);
642                 wcscat_s(buffer, 16383, line);
643
644                 SetDlgItemText(hWnd, IDC_EDIT5, buffer);
645             }
646             break;
647             case IDC_BUTTON_CLEAR:
648             {
649                 wcscpy_s(buffer, 16383, L"\0");
650                 SetDlgItemText(hWnd, IDC_EDIT5, buffer);
651             }
652             break;
653             case IDM_ABOUT:
654                 DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
655                 break;
```

```
656             case IDM_EXIT:
657                 DestroyWindow(hWnd);
658                 break;
659             default:
660                 return DefWindowProc(hWnd, message, wParam, lParam);
661             }
662         }
663         break;
664     case WM_PAINT:
665         {
666             PAINTSTRUCT ps;
667             HDC hdc = BeginPaint(hWnd, &ps);
668             // TODO: Add any drawing code that uses hdc here...
669             EndPaint(hWnd, &ps);
670         }
671         break;
672     case WM_SIZE:
673         if (hwndEdit5)
674         {
675             RECT rcClient;
676             GetClientRect(hWnd, &rcClient);
677             SetWindowPos(hwndEdit5, NULL, 10, 170, rcClient.right - 20,
678                 rcClient.bottom - 20, SWP_NOZORDER);
679         }
680         break;
681     case WM_DESTROY:
682         PostQuitMessage(0);
683         break;
684     default:
685         return DefWindowProc(hWnd, message, wParam, lParam);
686     }
687     return 0;
688 }
689
690 // Message handler for about box.
691 INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
692 {
693     UNREFERENCED_PARAMETER(lParam);
694     switch (message)
695     {
696     case WM_INITDIALOG:
697         return (INT_PTR)TRUE;
698
699     case WM_COMMAND:
700         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
701         {
702             EndDialog(hDlg, LOWORD(wParam));
703             return (INT_PTR)TRUE;
704         }
705         break;
706     }
707     return (INT_PTR)FALSE;
```

```
708  }
```