```cpp
// Commutator.cpp (c) Sunday, October 20, 2024
// by James Pate Williams, Jr., BA, BS, MSWE, PhD
// Exercise 1.2 in "Modern Quantum Chemistry
// Intrduction to Advanced Electronic Structure Theory" (c)
// 1996 by Attila Szabo and Neil S. Ostlund

#include <complex.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <iomanip>
#include <iostream>
using namespace std;

void SystemError(const char error_message[])
{
    cerr << error_message << endl;
    exit(-1);
}

long double** allocate_real_matrix(
    int lr, int ur, int lc, int uc)
{
    /*  Allocates a real matrix of range [lr..ur][lc..uc].  */

    int i;
    long double** p;

    p = (long double**)malloc((unsigned)(ur - lr + 1) * sizeof(long double*));
    if (!p) SystemError("Failure in allocate_real_matrix().");
    p -= lr;

    for (i = lr; i <= ur; i++) {
        p[i] = (long double*)malloc((unsigned)(uc - lc + 1) * sizeof(long
          double));
        if (!p[i]) SystemError("Failure in allocate_real_matrix().");
        p[i] -= lc;
    }
    return p;
}

long double* allocate_real_vector(int l, int u)
{
    /*  Allocates a real vector of range [l..u].  */

    long double* p;

    p = (long double*)malloc((unsigned)(u - l + 1) * sizeof(long double));
    if (!p) SystemError("Failure in allocate_real_vector().");
    return p - l;
}

```

```
 52 _Lcomplex** allocate_complex_matrix(
 53     int lr, int ur, int lc, int uc)
 54 {
 55     /*  Allocates a complex matrix of range [lr..ur][lc..uc].  */
 56 
 57     int i;
 58     _Lcomplex** p;
 59 
 60     p = (_Lcomplex**)malloc((unsigned)(ur - lr + 1) * sizeof(_Lcomplex*));
 61     if (!p) SystemError("Failure in allocate_real_matrix().");
 62     p -= lr;
 63 
 64     for (i = lr; i <= ur; i++) {
 65         p[i] = (_Lcomplex*)malloc((unsigned)(uc - lc + 1) * sizeof(_Lcomplex));
 66         if (!p[i]) SystemError("Failure in allocate_real_matrix().");
 67         p[i] -= lc;
 68     }
 69     return p;
 70 }
 71 
 72 void free_real_matrix(long double** m, int lr, int ur, int lc)
 73 {
 74     /*  Frees a real matrix of range [lr..ur][lc..uc].  */
 75 
 76     int i;
 77 
 78     for (i = ur; i >= lr; i--) free((char*)(m[i] + lc));
 79     free((char*)(m + lr));
 80 }
 81 
 82 void free_real_vector(long double* v, int l)
 83 {
 84     /*  Frees a real vector of range [l..u].  */
 85 
 86     free((char*)(v + l));
 87 }
 88 
 89 void free_complex_matrix(_Lcomplex** m, int lr, int ur, int lc)
 90 {
 91     /*  Frees a complex matrix of range [lr..ur][lc..uc].  */
 92 
 93     int i;
 94 
 95     for (i = ur; i >= lr; i--) free((char*)(m[i] + lc));
 96     free((char*)(m + lr));
 97 }
 98 
 99 void MatMul(
100     long double** A,
101     long double** B,
102     long double** C,
103     int m, int n, int p)
```

```
104 {
105     // (mxn)(nxp)=(mxp)
106 
107     for (int i = 1; i <= m; i++)
108     {
109         for (int j = 1; j <= n; j++)
110         {
111             double sum = 0.0;
112 
113             for (int k = 1; k <= n; k++)
114                 sum += A[i][k] * B[k][j];
115 
116             C[i][j] = sum;
117         }
118     }
119 }
120 
121 void MatAdd(
122     long double** A,
123     long double** B,
124     long double** C,
125     int m, int n, int s)
126 {
127     // (mxn)-(mxn)=(mxn)
128 
129     for (int i = 1; i <= m; i++)
130     {
131         for (int j = 1; j <= n; j++)
132             C[i][j] = A[i][j] - s * B[i][j];
133     }
134 }
135 
136 void Commutator(
137     long double** A,
138     long double** B,
139     long double** C,
140     int n)
141 {
142     // (mxn)(nxm) - (nxm)(mxn) = (mxm) - (nxn)
143     // therefore the matrices must be square (m=n)
144     long double** AB = allocate_real_matrix(1, n, 1, n);
145     long double** BA = allocate_real_matrix(1, n, 1, n);
146     MatMul(A, B, AB, n, n, n);
147     MatMul(B, A, BA, n, n, n);
148     MatAdd(AB, BA, C, n, n, -1);
149 }
150 
151 void AntiCommutator(
152     long double** A,
153     long double** B,
154     long double** C,
155     int n)
```

```
156 {
157     // (mxn)(nxm) - (nxm)(mxn) = (mxm) - (nxn)
158     // therefore the matrices must be square (m=n)
159     long double** AB = allocate_real_matrix(1, n, 1, n);
160     long double** BA = allocate_real_matrix(1, n, 1, n);
161     MatMul(A, B, AB, n, n, n);
162     MatMul(B, A, BA, n, n, n);
163     MatAdd(AB, BA, C, n, n, 1);
164 }
165
166 void Adjoint(
167     _Lcomplex** cA,
168     _Lcomplex** cB,
169     int n, int m)
170 {
171     for (int i = 1; i <= n; i++)
172     {
173         for (int j = 1; j <= m; j++)
174             cB[j][i] = conj(cA[i][j]);
175     }
176 }
177
178 _Lcomplex cTrace(_Lcomplex** M, int n)
179 {
180     _Lcomplex trace = _LCbuild(0, 0);
181
182     for (int i = 1; i <= n; i++)
183     {
184         trace._Val[0] += M[i][i]._Val[0];
185         trace._Val[1] += M[i][i]._Val[1];
186     }
187
188     return trace;
189 }
190
191 void cMatMul(
192     _Lcomplex** cA,
193     _Lcomplex** cB,
194     _Lcomplex** cC,
195     int n, int m, int p)
196 {
197     // (nxm)(mxp)=(nxp)
198
199     for (int i = 1; i <= n; i++)
200     {
201         for (int j = 1; j <= p; j++)
202         {
203             _Lcomplex csum = { };
204
205             for (int k = 1; k <= m; k++)
206             {
207                 _Lcomplex product =
```

```
208                     _LCmulcc(cA[i][k], cB[k][j]);
209                 csum._Val[0] += product._Val[0];
210                 csum._Val[1] += product._Val[1];
211             }
212
213             cC[i][j]._Val[0] = csum._Val[0];
214             cC[i][j]._Val[1] = csum._Val[1];
215         }
216     }
217 }
218
219 void PrintMatrix(
220     const char title[], long double** M, int m, int n,
221         ostream& ofile)
222 {
223     char line[128] = { };
224     int cchMax = 127;
225     ofile << title << endl << endl;
226
227     for (int i = 1; i <= m; i++)
228     {
229         for (int j = 1; j <= n; j++)
230         {
231             sprintf_s(line, "%+4.0Lf  ", M[i][j]);
232             ofile << line;
233         }
234
235         ofile << endl;
236     }
237
238     ofile << endl;
239 }
240
241 void cPrintMatrix(
242     const char title[], _Lcomplex ** M, int m, int n,
243         ostream& ofile)
244 {
245     char line[128] = { };
246     int cchMax = 127;
247     ofile << title << endl << endl;
248
249     for (int i = 1; i <= m; i++)
250     {
251         for (int j = 1; j <= n; j++)
252         {
253             sprintf_s(line, "%+13.6Lf  ", M[i][j]._Val[0]);
254             ofile << line;
255             sprintf_s(line, "%+13.6Lf  ", M[i][j]._Val[1]);
256             ofile << line << endl;
257         }
258
259         ofile << endl;
```

```
260     }
261
262     ofile << endl;
263 }
264
265 _Lcomplex** CreateComplexMatrix(
266     int n, int m)
267 {
268     _Lcomplex** C = allocate_complex_matrix(1, n, 1, m);
269
270     for (int i = 1; i <= n; i++)
271     {
272         for (int j = 1; j <= m; j++)
273         {
274             long double real = rand() % 10;
275             long double imag = rand() % 10;
276             C[i][j] = _LCbuild(real, imag);
277         }
278     }
279
280     return C;
281 }
282
283 _Lcomplex** CreateUnitaryMatrix()
284 {
285     long double pi = 4.0 * atanl(1.0);
286     _Lcomplex** U = allocate_complex_matrix(1, 2, 1, 2);
287     U[1][1] = _LCbuild(1.0 / sqrtl(2.0), 0.0);
288     U[1][2] = _LCbuild(0.0, -1.0 / sqrtl(2.0));
289     U[2][1] = _LCbuild(1.0 / sqrtl(2.0), 0.0);
290     U[2][2] = _LCbuild(0.0, 1.0 / sqrtl(2.0));
291     return U;
292 }
293
294 void Inverse2x2(long double** A, long double** AI)
295 {
296     long double det = A[1][1] * A[2][2] - A[1][2] * A[2][1];
297
298     AI[1][1] = +A[2][2];
299     AI[2][2] = +A[1][1];
300     AI[1][2] = -A[1][2];
301     AI[2][1] = -A[2][1];
302
303     for (int i = 1; i <= 2; i++)
304         for (int j = 1; j <= 2; j++)
305             AI[i][j] = A[i][j] / det;
306 }
307
308 int main()
309 {
310     int m = 2, n = 3, p = 4;
311     long double A33[4][4] = {
```

```
312         { 0, 0, 0, 0},
313         { 0, 1, 1, 0 },
314         { 0, 1, 2, 2 },
315         { 0, 0, 2, -1 } };
316     long double B33[4][4] = {
317         { 0, 0, 0, 0 },
318         { 0, 1, -1, 1 },
319         { 0, -1, 0, 0 },
320         { 0, 1, 0, 1} };
321     long double D22[3][3] = {
322         { 0, 0, 0 },
323         { 0, 1, 2 },
324         { 0, 3, 4 }
325     };
326     long double E22[3][3] = {
327         { 0, 0, 0 },
328         { 0, 5, 6 },
329         { 0, 7, 8 }
330     };
331     long double** A = allocate_real_matrix(1, n, 1, n);
332     long double** B = allocate_real_matrix(1, n, 1, n);
333     long double** AB = allocate_real_matrix(1, n, 1, n);
334     long double** BA = allocate_real_matrix(1, n, 1, n);
335     long double** D = allocate_real_matrix(1, 2, 1, 2);
336     long double** E = allocate_real_matrix(1, 2, 1, 2);
337     long double** DE = allocate_real_matrix(1, 2, 1, 2);
338     long double** ED = allocate_real_matrix(1, 2, 1, 2);
339     long double** DI = allocate_real_matrix(1, 2, 1, 2);
340     long double** EI = allocate_real_matrix(1, 2, 1, 2);
341     long double** DEI = allocate_real_matrix(1, 2, 1, 2);
342     long double** DIEI = allocate_real_matrix(1, 2, 1, 2);
343     unsigned int s = rand() % 25 + 1;
344     _Lcomplex** cA = CreateComplexMatrix(n, m);
345     _Lcomplex** cB = CreateComplexMatrix(m, p);
346     _Lcomplex** cU = CreateUnitaryMatrix();
347     _Lcomplex** cOmega = CreateComplexMatrix(n, n);
348     _Lcomplex** cAj = allocate_complex_matrix(1, m, 1, n);
349     _Lcomplex** cBj = allocate_complex_matrix(1, p, 1, m);
350     _Lcomplex** cUj = allocate_complex_matrix(1, 2, 1, 2);
351     _Lcomplex** cAB = allocate_complex_matrix(1, n, 1, p);
352     _Lcomplex** cABj = allocate_complex_matrix(1, p, 1, n);
353     _Lcomplex** cBjAj = allocate_complex_matrix(1, p, 1, n);
354     _Lcomplex** cUjOmega = allocate_complex_matrix(1, 2, 1, 2);
355     _Lcomplex** cUjOmegaU = allocate_complex_matrix(1, 2, 1, 2);
356     _Lcomplex** cUUj = allocate_complex_matrix(1, 2, 1, 2);
357
358     ofstream ofile;
359
360     ofile.open("CommumtatorResults.txt",
361         ofstream::ate | ostream::trunc);
362
363     srand(s);
```

```
364
365     for (int i = 1; i <= n; i++)
366         for (int j = 1; j <= n; j++)
367             A[i][j] = A33[i][j];
368
369     for (int i = 1; i <= n; i++)
370         for (int j = 1; j <= n; j++)
371             B[i][j] = B33[i][j];
372
373     PrintMatrix("Matrix A:", A, n, n, ofile);
374     PrintMatrix("Matrix B:", B, n, n, ofile);
375     Commutator(A, B, AB, n);
376     PrintMatrix(
377         "Commutator:", AB, n, n, ofile);
378     AntiCommutator(A, B, AB, n);
379     PrintMatrix(
380         "Anti-commutator:", AB, n, n, ofile);
381     cMatMul(cA, cB, cAB, n, m, p);
382     Adjoint(cA, cAj, n, m);
383     Adjoint(cB, cBj, m, p);
384     Adjoint(cAB, cABj, n, p);
385     cMatMul(cBj, cAj, cBjAj, p, m, n);
386     cPrintMatrix("Matrix A:", cA, n, m, ofile);
387     cPrintMatrix("Matrix B:", cB, m, p, ofile);
388     cPrintMatrix("Matrix AB:", cAB, n, p, ofile);
389     cPrintMatrix("Matrix A-Adjoint:", cAj, m, n, ofile);
390     cPrintMatrix("Matrix B-Adjoint:", cBj, p, m, ofile);
391     cPrintMatrix("Matrix AB-Adjoint:", cABj, p, n, ofile);
392     cPrintMatrix("Matrix B-Adjoint A-Adjoint:", cBjAj, p, n, ofile);
393     bool equal = true;
394
395     for (int i = 1; equal && i <= n; i++)
396     {
397         for (int j = 1; equal && j <= m; j++)
398         {
399             bool ereal = fabsl(cABj[i][j]._Val[0] -
400                 cBjAj[i][j]._Val[0]) < 1.0e-10;
401             bool eimag = fabsl(cABj[i][j]._Val[1] -
402                 cBjAj[i][j]._Val[1]) < 1.0e-10;
403             equal = ereal && eimag;
404         }
405     }
406
407     if (equal)
408         ofile << "Complex adjoints are equal" << endl << endl;
409     else
410         ofile << "Complex adjoints are not equal" << endl << endl;
411
412     ofile << "Example 1.8\n\n";
413     cU = CreateUnitaryMatrix();
414     Adjoint(cU, cUj, 2, 2);
415     cMatMul(cUj, cOmega, cUjOmega, 2, 2, 2);
```

```
416     cMatMul(cU, cUj, cUUj, 2, 2, 2);
417     cMatMul(cUjOmega, cU, cUjOmegaU, 2, 2, 2);
418     cPrintMatrix("Matrix U:", cU, 2, 2, ofile);
419     cPrintMatrix("Matrix U-Adjoint:", cUj, 2, 2, ofile);
420     cPrintMatrix("Matrix U U-Adjoint:", cUUj, 2, 2, ofile);
421     cPrintMatrix("Matrix Omega:", cOmega, 2, 2, ofile);
422     cPrintMatrix("Matrix U-Adjoint Omega U:", cUjOmegaU, 2, 2, ofile);
423
424     ofile << "Trace Omega.Real        = ";
425     ofile << cTrace(cOmega, 2)._Val[0] << endl;
426     ofile << "Trace Omega.Imag        = ";
427     ofile << cTrace(cOmega, 2)._Val[1] << endl;
428     ofile << "Trace U-a Omega U.Real  = ";
429     ofile << cTrace(cUjOmegaU, 2)._Val[0] << endl;
430     ofile << "Trace U-a Omega U.Imag  = ";
431     ofile << cTrace(cUjOmegaU, 2)._Val[1] << endl;
432     ofile << endl;
433
434     for (int i = 1; i <= 2; i++)
435         for (int j = 1; j <= 2; j++)
436             D[i][j] = D22[i][j];
437
438     for (int i = 1; i <= 2; i++)
439         for (int j = 1; j <= 2; j++)
440             E[i][j] = E22[i][j];
441
442     Inverse2x2(D, DI);
443     Inverse2x2(E, EI);
444     MatMul(D, E, DE, 2, 2, 2);
445     Inverse2x2(DE, DEI);
446     MatMul(DI, EI, DIEI, 2, 2, 2);
447
448     ofile << "Exercise 1.4 b.\n\n";
449     PrintMatrix("Matrix D:", D, 2, 2, ofile);
450     PrintMatrix("Matrix E:", E, 2, 2, ofile);
451     PrintMatrix("Matrix D-Inverse:", DI, 2, 2, ofile);
452     PrintMatrix("Matrix E-Inverse:", EI, 2, 2, ofile);
453     PrintMatrix("Matrix D-Inverse Times E-Inverse:",
454         DIEI, 2, 2, ofile);
455     PrintMatrix("Matrix DE-Inverse:", DEI, 2, 2, ofile);
456
457     ofile.close();
458
459     free_real_matrix(A, 1, n, 1);
460     free_real_matrix(B, 1, n, 1);
461     free_real_matrix(AB, 1, n, 1);
462     free_real_matrix(BA, 1, n, 1);
463     free_real_matrix(D, 1, 2, 1);
464     free_real_matrix(E, 1, 2, 1);
465     free_real_matrix(DE, 1, 2, 1);
466     free_real_matrix(ED, 1, 2, 1);
467     free_real_matrix(DI, 1, 2, 1);
```

```
468     free_real_matrix(EI, 1, 2, 1);
469     free_real_matrix(DEI, 1, 2, 1);
470     free_real_matrix(DIEI, 1, 2, 1);
471     free_complex_matrix(cA, 1, n, 1);
472     free_complex_matrix(cB, 1, m, 1);
473     free_complex_matrix(cAj, 1, m, 1);
474     free_complex_matrix(cBj, 1, p, 1);
475     free_complex_matrix(cUj, 1, 2, 1);
476     free_complex_matrix(cAB, 1, n, 1);
477     free_complex_matrix(cABj, 1, m, 1);
478     free_complex_matrix(cBjAj, 1, m, 1);
479     free_complex_matrix(cUUj, 1, 2, 1);
480     free_complex_matrix(cOmega, 1, 2, 1);
481     free_complex_matrix(cUjOmega, 1, 2, 1);
482     free_complex_matrix(cUjOmegaU, 1, 2, 1);
483 }
```