ECE 2036 Lab #5 – Complex Matrix Class

Due Date: 5:00 pm - Tuesday, November 11th (both sections)

1. Overview

The goal of this lab is to practice operator overloading (big time), as well as dynamic memory allocation, class composition and creating custom destructor classes. We will be making two classes (`Complex` and `Matrix`), in order to support operations involving complex numbers, and operations involving complex matrices.

2. Complex Number Class (50%)

As you hopefully recall from a Trigonometry (or Pre-Calculus) course, complex numbers are numbers that have both a real part, and an imaginary part. They are typically written in the form:

$$a + bi \text{ , (where } i \equiv \sqrt{-1} \text{ ; a \& b are both real numbers)}$$

For this lab, we are going to write our own complex number class (`Complex`), which will overload quite a few operators to make it behave as we would expect a complex number to. It needs to have two private data members (`double real` and `double imaginary`). It should have a default constructor, which sets both data members to zero, as well as a constructor that can take both real and imaginary coefficients directly).

```
Complex(); // creates 0+0i
Complex(2, -1.5); // creates 2-1.5i
```

You also need to provide standard getters/settings for the two required data members.

You can include any other methods or data members you need/want to support the required behaviors for the class, but it must have the constructors and getters/setters specified above, and it must support all of the operations listed in section **2.1**.

2.1 Overloaded Operators for `Complex` Class

`Complex + Complex` – this needs to return a complex number where the real parts have been added together, and the imaginary parts have been added together.

`Complex + double` (`double + Complex`) – this needs to return a complex number where the real part has had the value provided added to it (no change to the imaginary part). This must be a symmetric operator.

`Complex == Complex` – returns true if both the real parts and the imaginary parts of both complex numbers are equal, otherwise it returns false.

`Complex += Complex` – same as `Complex + Complex` above, but it modifies the original Complex instance on the left-hand-side and returns a reference to it.

`Complex += double` – same as `Complex + double` above, but it modifies the original Complex instance on the left-hand-side and returns a reference to it.

`-Complex` – this returns a Complex number with both the real and imaginary parts multiplied by `-1`.

`Complex - Complex` – same as `Complex + Complex`, but with subtraction instead of addition.

`Complex - double` – returns `Complex + (-double)`.
`double - Complex` – return `double + (-Complex)`.

`Complex * Complex` – for two complex numbers, a + bi and c + di, this returns the new complex number given by the formula ($ac - db + (bc + ad)i$). This is the standard FOIL result for binary polynomials.

`Complex * double (double * Complex)` – this returns a new complex number where both the real and imaginary parts have been multiplied by the supplied double.

`Complex / double` – this returns a new complex number where both the real and imaginary parts have been divided by the supplied double.

`Complex / Complex` – this is the trickiest mathematical operator for complex numbers, since it has to be converted to a form that can still be represented as
x + y**i*. It needs to return a new complex number with the real and imaginary parts determined by the equation below (the final line):

$$(a+bi)/(c+di) = a/(c+di) + bi/(c+di)$$
$$= \frac{a(c-di)}{(c+di)(c-di)} + \frac{bi(c-di)}{(c+di)(c-di)}$$
$$= \frac{(ac-adi)}{(c^2+d^2)} + \frac{(bic+bd)}{(c^2+d^2)}$$
$$= \frac{(ac+bd)}{(c^2+d^2)} + \frac{(bc-ad)}{(c^2+d^2)}i$$

`double / Complex` – this returns a new complex number. It is essentially a special case of `Complex / Complex` (where b is zero). Support this either through an implicit cast, or via a dedicated overloaded operator function.

`~Complex` – this returns a new complex number that is the complex conjugate of the original complex number (the imaginary portion has been multiplied by `-1`)

Additionally, the Complex class needs overloaded versions of the >> and << operators that support the following actions:

```
Complex c1;

cout << "Enter a complex number (ex: 1.2+5.3i - no spaces!):\n";
cin >> c1;

cout << "The complex number you entered was:" << c1 << endl;
// prints: 1.2+5.3i, for example
```

The stdlib `atof()` function (which takes a C-string as input to return a double) will probably be of interest here. If either the real or imaginary portions of the string are zero, printing that number should not include the part that is zero (ex: 0+5.1i should print as just "5.1i", 1+0i should just print as "1"). You do not have to provide support for implied coefficients on user input (`1-i`, `2+i`, etc.). If both coefficients are zero, you should print "0". **You do need to make sure you can support negative coefficients for**

either the real or imaginary parts of the input number! If you support input from `cin` like `-5i`, `7.05`, or `1-i` properly, that will be worth some extra credit.

3. Complex Matrix Class (50%)

Now we are going to build upon our `Complex` class to make a complex `Matrix` class. If you remember from linear algebra, calculations involving matrices are fairly non-trivial (especially multiplication and division). In order to simplify some of this, we're going to implement our `Matrix` class to strictly support square matrices.

The `Matrix` class must have a default constructor (which creates a 1x1 matrix with the value of 0+0i as its single member). It must also have a constructor that takes in a single unsigned integer, which creates a square matrix of the provided size (throw the `invalid_argument` exception if they try to make a 0x0 matrix) where all the members are initially 0+0i. You should also create a constructor that takes an integer size, as well as a reference to an std::vector of Complex numbers, which you will use to fill the Matrix terms in by copying the provided values (from the top-left corner, line-by-line, from left to right) here, you can assume the size matches the length of the array (3x3 needs 9 terms, for instance).

```
std::array<Complex, 4> v0_init = {Complex(-1,0), Complex(3,4), Complex(0,1),
  Complex(1,0)};
vector<Complex> mat_vals(v0_init.begin(), v0_init.end());

Matrix();  // 1x1 matrix with single member: 0+0i
Matrix(0);  // throws invalid argument
Matrix(3);  // creates a 3x3 matrix with 0+0i as all 9 terms
Matrix(2, mat_vals);  // creates a 2x2 matrix that looks like:
                 // -1   3+4i
                 //  i   1
```

Importantly, **your matrix class *must* use dynamic memory allocation** to create locations in memory to store all of the required `Complex` members inside. This means you'll need your own custom destructor method to avoid memory leaks.

Your `Matrix` class should also have a `.size()` method that returns the current size, and a `.size(unsigned int)` method that sets the size to a new value (keeping any data members that fit into the new size – the top-left corner stays exactly where it was). Resizing to a larger matrix should default all new terms to 0+0i. Don't forget to throw the `invalid_argument` exception if they try to set the size to zero this way.

As with the `Complex` class, you can include any other methods or data members you need/want to support the required behaviors for the class, but it must have the constructors and getters/setters specified above, and it must support all of the operations listed in section **3.1**.

3.1 Overloaded Operators for `Matrix` Class

`mat1(unsigned int row, unsigned int col)` – this requires overloading the function operator, (), so you can perform indexing into a matrix instance (returning references to the value inside) allowing assignment an value reading. This method needs to perform bounds checking and throw the

`out_of_range` exception if used to index outside the Matrix domain/range. Like arrays, the top-left corner element of a Matrix is referenced as mat1(0,0).

`Matrix + Matrix` – this needs to return a new matrix where the terms of the original matrices have been added together term-wise (this is normal matrix addition).

`Matrix = Matrix` – this must be overridden from the default behavior because our Matrix class is performing dynamic memory allocation internally. You need to explicitly copy all the terms of the r.h.s. matrix into the l.h.s. matrix. Any matrix can be assigned to any other matrix without restriction.

`Matrix == Matrix` – this returns true or false by comparing terms of each matrix (and the size).

`Matrix += Matrix` – same as `Matrix + Matrix` above, but it modifies the original `Matrix` instance on the left-hand-side and returns a reference to it.

`-Matrix` – this returns a new `Matrix` where every term has been multiplied by `-1`.

`Matrix - Matrix` – same as `Matrix + Matrix`, but with subtraction instead of addition.

`Matrix * Matrix` – returns a new matrix with the terms determined by matrix multiplication rules (Wikipedia link):

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \, .$$

`Matrix * Complex (Complex * Matrix)` – this returns a new `Matrix` where every term has been multiplied by the `Complex` number provided.

`Matrix * double (double * Matrix)` – this returns a new `Matrix` where every term has been multiplied by the `double` provided (this is a special case of Matrix * Complex, and can be handled however you choose – similar to `double / Complex` above).

`Matrix / Complex` – this returns a new matrix where every term has been divided by the `Complex` number provided.

`Matrix / double` – this returns a new matrix where every term has been divided by the double provided (again, this is a special case of `Matrix / Complex`).

`~Matrix` – (not the destructor!) this returns a new matrix where every term is the complex conjugate of the original term.

For the following operators involving two matrices (+, -, *, +=), each of these overloaded uses of the operator should throw a `domain_error` if the size of the two matrices involved does not match. This might be best accomplished through some utility function.

Additionally, the Matrix class needs overloaded versions of the >> and << operators that support the following actions:

```
Matrix mat1;
unsigned int user_size;
```

```
cout << "How large do you wish to make your square matrix?:\n";
cin >> user_size;
mat1.size(size)
cout << "Please enter the " << (user_size * user_size)
     << " terms required to create your matrix on a single line "
     << "(ex: 1+2i 1+0i 0-5i 3.2+5.25i):\n";
cin >> mat1; // here, if they don't provide enough numbers, set
             // all remaining terms to 0+0i.

cout << "The matrix you entered was:\n" << mat1 << endl;
// prints (ugly, though it may be with 3 spaces between each term):
//   1+2i   1
//   -5i   3.2+5.25i
```

4. Sample Test Program

There honestly may be a few bugs in this section – they will be corrected ASAP.

If this program runs and returns the expected output from the given inputs, you should expect a fairly decent grade (assuming you didn't just hard-code the exact values provided):

4.1 Sample Program Code

```
#include <array>
#include <exception>
#include <vector>
#include "Complex.h"
#include "Matrix.h"

using namespace std;

int main() {
  Complex c1;
  Complex c2;
  Complex c3(1,1);

  cout << "Testing Complex Class:" << endl;
  cout << "c1, c2, c3: " << c1 << ", " << c2 << ", " << c3 << endl;

  cout << "\nPlease enter values for c1, c2 (ex: 1+5.3i 1.2-i):" << endl;
  cin >> c1 >> c2; // the sample output is for the example input

  cout << "c1, c2, c3: " << c1 << ", " << c2 << ", " << c3 << endl;
  cout << "\nThe real parts of c1, c2, c3: " << c1.getReal() << ", " << c2.getReal()
       << ", " << c3.getReal() << endl;

  cout << "\n~c3: " << (~c3) << endl;
  cout << "c1 + c3: " << (c1 + c3) << endl;

  cout << "\nc1 - c3: " << (c1 - c3) << endl;
  cout << "c1 += (-c3): " << (c1 += (-c3)) << endl;
  cout << "c1 / c3: " << (c1 / c3) << endl;

  cout << "\n2 * c1 + 7: " << (2 * c1 + 7) << endl;
  cout << "3 / c3: " << (3 / c3) << endl;
  cout << "c2 * c3: " << (c2 * c3) << endl;
```

```
cout << "\nSetting c1 to c3..." << endl;
c1 = c3;
cout << "c1 == c3: " << (c1 == c3 ? "true" : "false") << endl;
cout << "c2 == c3: " << (c2 == c3 ? "true" : "false") << endl;


// because we can't rely on C++11 vector list initialization...
array<Complex, 4> v0_init = {Complex(-1,0), Complex(3,4), Complex(0,1), Complex(1,0)};
vector<Complex> mat_vals(v0_init.begin(), v0_init.end());

Matrix mat1;
Matrix mat2(2, mat_vals);
Matrix mat3(2);

cout << "\n\nTesting Matrix Class:" << endl;
cout << "mat1:\n" << mat1 << endl;
cout << "mat2:\n" << mat2 << endl;
cout << "mat3:\n" << mat3 << endl;

try {
  cout << "\nChanging mat2 to a 0x0 matrix..." << endl;
  mat2 = Matrix(0);
} catch (const invalid_argument& e) {
  cerr << "Error Caught: " << e.what() << endl;
}

cout << "\nChanging mat2 to a 3x3 matrix..." << endl;
mat2.size(3);
cout << "mat2 is now the following " << mat2.size() << "x" << mat2.size() << " matrix:" << endl;
cout << mat2 << endl;

cout << "\nChanging mat2 back to a 2x2 matrix..." << endl;
mat2.size(2);
cout << "mat2 is now the following " << mat2.size() << "x" << mat2.size() << " matrix:" << endl;
cout << mat2 << endl;

cout << "\nSetting mat3(1,1) to 7+2i..." << endl;
mat3(1,1) = Complex(7,2);
cout << "mat3 is now:" << endl;
cout << mat3 << endl;

try {
  cout << "\nAccessing mat3(2,1)..." << endl;
  mat3(2,1) = Complex(7.1, 0);
} catch (const out_of_range& e) {
  cerr << "Error Caught: " << e.what() << endl;
}

Matrix mat4;
unsigned int user_size;
cout << "\nTesting user input:" << endl;
cout << "How large would you like to make your matrix?: ";
cin >> user_size;  // 2 was entered for the sample output
mat4.size(user_size);

cout << "Please enter the " << (user_size * user_size)
     << " terms required to create your matrix on a single line "
     << "\n(ex: 1+2i 1+0i 0-5i 3.2+5.25i):\n";
cin >> mat1; // here, the sample output user entered the first 3 sample numbers (they forgot 1)
cout << "The matrix you entered was:\n" << mat4 << endl;
```

```
  cout << "\nMatrix Math:" << endl;
  cout << "~mat2:\n" << (~mat2) << endl;
  try {
    cout << "\nTrying to add mat1 to mat3..." << endl;
    cout << "mat1 + mat3:" << endl << (mat1 + mat3) << endl;
  } catch (const domain_error& e) {
    cerr << "Error Caught: " << e.what() << endl;
  }

  cout << "\nmat2 + mat3:\n" << (mat2 + mat3) << endl;
  cout << "\nmat2 - mat3:\n" << (mat2 - mat3) << endl;
  cout << "\nmat3 += (-mat2):\n" << (mat3 += (-mat2)) << endl;
  cout << "\nmat3 / mat3(1,0):\n" << (mat3 / mat3(1,0)) << endl;

  cout << "\n2.0 * mat2:\n" << (2.0 * mat2) << endl;

  cout << "\nmat3 * mat2 / 2.0:\n" << (mat3 * mat2 / 2.0) << endl;

  cout << "\nSetting mat1 to mat3..." << endl;
  mat1 = mat3;
  cout << "mat1 == mat3: " << (mat1 == mat3 ? "true" : "false") << endl;
  cout << "mat2 == mat3: " << (mat2 == mat3 ? "true" : "false") << endl;
}
```

4.2 Sample Program Output

```
Testing Complex Class:
c1, c2, c3: 0, 0, 1+i

Please enter values for c1, c2 (ex: 1+5.3i 1.2-i):
1+5.3i 1.2-1i
c1, c2, c3: 1+5.3i, 1.2-i, 1+i

The real parts of c1, c2, c3: 1, 1.2, 1

~c3: 1-i
c1 + c3: 2+6.3i

c1 - c3: 4.3i
c1 += (-c3): 4.3i
c1 / c3: 2.15+2.15i

2 * c1 + 7: 7+8.6i
3 / c3: 1.5-1.5i
c2 * c3: 2.2+0.2i

Setting c1 to c3...
c1 == c3: true
c2 == c3: false


Testing Matrix Class:
mat1:
0

mat2:
-1   3+4i
i   1

mat3:
```

```
0   0
0   0


Changing mat2 to a 0x0 matrix...
Error Caught: Matrix size must be at least 1.

Changing mat2 to a 3x3 matrix...
mat2 is now the following 3x3 matrix:
-1   3+4i   0
i   1   0
0   0   0


Changing mat2 back to a 2x2 matrix...
mat2 is now the following 2x2 matrix:
-1   3+4i
i   1


Setting mat3(1,1) to 7+2i...
mat3 is now:
0   0
0   7+2i


Accessing mat3(2,1)...
Error Caught: Index (2,1) out of 2x2 matrix bounds.

Testing user input:
How large would you like to make your matrix?: 2
Please enter the 4 terms required to create your matrix on a single line
(ex: 1+2i 1+0i 0-5i 3.2+5.25i):
1+2i 1+0i 0-5i
The matrix you entered was:
1+2i   1
-5i   0


Matrix Math:
~mat2:
-1   3-4i
-i   1


Trying to add mat1 to mat3...
mat1 + mat3:
Error Caught: Adding matrices with non-matching sizes is not allowed.

mat2 + mat3:
-1   3+4i
i   8+2i


mat2 - mat3:
-1   3+4i
i   -6-2i


mat3 += (-mat2):
1   -3-4i
```

```
-i   6+2i


mat3 / mat3(1,0):
i   4-3i
1   -2+6i


2.0 * mat2:
-2   6+8i
2i   2


mat3 * mat2 / 2.0:
1.5-1.5i   0
-1+3.5i   5-0.5i


Setting mat1 to mat3...
mat1 == mat3: true
mat2 == mat3: false
```

5. Assignment Submission

Submit the assignment under Project 5 / Lab 5 on T-Square. Your Matrix and Complex class implementations must be in separate files from one another, but you should implement each entirely in its .h file (make sure you prevent double-inclusion of Complex.h in any program which uses both header files!). Make sure your header files work with the provided sample program. The TAs will grade your projects directly from your T-Square submissions (which are due by 5pm on the due date), so make sure you've included all necessary files.

Even though you are not demoing directly to the TAs, the GT Honor Code is still very much in effect, so you must submit solutions that you personally created (you can still work together on parts of the problem together, but you have to type in your own solutions personally).

6. Extra Credit

+2 for Complex being able to handle any legitimate input from cin (0, 3i, -7.1, -.001i, etc.)
+2 for prettier printing of cout << mat1. May sure all the outputs are aligned well; rounding to 2 decimal points is acceptable.
+3 for supporting `Matrix` division. You'll need to be able to calculate inverse matrices, which involves determinants and transpose operations. Note that determinants are somewhat involved with matrices over 3x3.