1 Objectives

1. To implement an abstract data type (ADT)[1]
2. To integrate an ADT into the C++ language using the operator overloading facility of the C++ language
3. To learn about function objects and how to define them

2 Assignment Background

An abstract data type (ADT) represents a set of values with associated operations, paying no attention to details of representation of data and implementation of operations. Classic ADTs such as rational number and complex number ADTs support many arithmetic operations, making them ideal data types for operator overloading.

However, a Google search for "class rational c++" reveals many ready to go C++ classes for rational numbers; same is true for fraction and complex number ADTs. As a result, an assignment designed to provide practice with operator overloading must stay away form those classic ADTs; otherwise, some students might miss out on getting a real feel for the concept.

Therefore, in this assignment, we are going to create an ADT that not only is not as ubiquitous as rational and complex number ADTs but lends itself to operator overloading just as well.

3 Introducing ADT Mat2x2

Mat2x2 is an abstract data type representing 2×2 matrices of the form $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ where the entries a, b, c, and d are all real numbers. For example, the matrix $X = \begin{pmatrix} 4 & 7 \\ 2 & 6 \end{pmatrix}$ represents a value of type **Mat2x2**. The value $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ represents the identity matrix of type **Mat2x2**.

The Arithmetic operations on **Mat2x2** objects such as $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $M' = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}$ are listed below, where x and the entries of M and M' are all real numbers.

[1] ADT = Values + Operations − Implementation details

Notation	$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $M' = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}$, and x is a number
Operation	Definition
Addition **Subtraction**	$M \pm M' = \begin{pmatrix} a \pm a' & b \pm b' \\ c \pm c' & d \pm d' \end{pmatrix}$
Scalar Addition **Scalar Subtraction**	$M \pm x = \begin{pmatrix} x \pm a & x \pm b \\ x \pm c & x \pm d \end{pmatrix}$ and $x \pm M = \pm(M \pm x)$
Multiplication	$M * M' = \begin{pmatrix} aa' + bc' & ab' + bd' \\ ca' + dc' & cb' + dd' \end{pmatrix}$
Scalar Multiplication	$M * x = \begin{pmatrix} xa & xb \\ xc & xd \end{pmatrix} = x * M$
Inversion	$M^{-1} = \dfrac{1}{ad - bc} * \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$ provided that $ad - bc \neq 0$
Division	$M/M' = M * M'^{-1}$
Scalar Division	$M/x = M * \left(\dfrac{1}{x}\right)$ provided that $x \neq 0$
Scalar Division	$x/M = x * M^{-1}$
Transpose	$M^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$
Determinant	$\text{det}(M) = ad - bc$
Trace	$\text{tr}(M) = a + d$
Eigenvalues	$\lambda_1, \lambda_2 = \begin{cases} \dfrac{\text{tr}(M) \pm \sqrt{\Delta}}{2} & \text{if } \Delta >= 0 \\ \left(\dfrac{\text{tr}(M)}{2}, \pm\dfrac{\sqrt{-\Delta}}{2}\right) & \text{if } \Delta < 0 \end{cases}$ where $\Delta = (\text{tr}(M))^2 - 4(\text{det}(M))$
Symmetric?	M is symmetric if $b = c$.
Equal?	$M = M'$ if $\lvert a - a' \rvert < \epsilon$, $\lvert b - b' \rvert < \epsilon$, $\lvert c - c' \rvert < \epsilon$, and $\lvert d - d' \rvert < \epsilon$, where ϵ =1.0E-6
Similar?	M is similar to M' if $\text{det}(M) = \text{det}(M')$ and $\text{tr}(M) = \text{tr}(M')$

4 Your Task

Implement the **Mat2x2** ADT described above. Your **Mat2x2** class should have the following members:

- Default constructor: sets all four entries to zero.
- Normal constructor: accepts initial values for all four entries as parameters.
- Default copy constructor, default assignment operator, default destructor
- Overloaded input operator for reading **Mat2x2** values
- Overloaded output operator for writing **Mat2x2** values
- **determinant()**, **trace()**, **isSymmetric()**, **isSimilar()**, **transpose()**, **inverse()**, throwing `std::overflow_error("Inverse undefined");` if $|\text{det}(M)| \le \epsilon$, where ϵ =1.0E-6.

  Also, for scalar division `throw std::overflow_error("Division by zero");` if $|x| \le \epsilon$.

Provide the following operations on **Mat2x2** objects M and M', and a real number x:

Compound assignments `M+=M’, M-=M’, M*=M’, M/=M’`, `M+=x, M-=x, M*=x, M/=x`

Basic arithmetic `M+M’, M-M’, M*M’, M/M’`, `M+x, M-x, M*x, M/x`, `x+M, x-M, x*M, x/M`

Relational `M==M, M!=M’`

pre/post-increment `++M, M++`

pre/post-decrement `--M, M--`

Subscripts Both const and non-const versions. The subscripts and the matrix entries should be associated as follows: $\begin{pmatrix} [0] & [1] \\ [2] & [3] \end{pmatrix}$. If subscript is invalid, must throw: `invalid_argument("index out of bounds")`

Function objects

- M(int) returns an eigenvalue as a `vector<double>`. Valid calls:
  - M(1) returns λ_1 as a `vector<double>`
  - M(2) returns λ_2 as a `vector<double>`

  Must throw `invalid_argument("invalid argument")` if the parameter value is invalid.

  Note that the size of the vectors returned by the calls $M(1)$ and $M(2)$ is either 1 or 2. Specifically, the size of the returned vectors is 1 for real roots and 2 for complex roots; that's obviously because your code would call the vector's **push_back()** only once for real roots, but would call **push_back()** twice for complex roots, where the first call stores the real part and the second call stores the imaginary part.
- M() returns det(M)

5 Basic guidelines

Use the following guidelines[2] for choosing between overloading an operator as member or non-member:

Operator	Recommended Implementation
=, ( ), [ ], − >	must be member
All unary operators	member
Compound assignment operators	member
All other binary operators	non-member

6 Sample Test Driver

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cassert>
#include "Matrix2x2.h"
using namespace std;

/*
Tests class Matrix2x2. Specifically, tests constructors, compound assignment
operator overloads, basic arithmetic operator overloads, unary +, unary -,
pre/post-increment/decrement, subscripts, function objects, eigenvalues,
input/output operators, isSymetric, isSimilar, determinant, trace, and
equality relational operators.

The test starts with matrix

|2.00 -1.00|
|          |
|1.00  2.00|

with these two complex eigenvalues:
root 1: 2 +1i
root 2: 2 -1i

@return 0 to indicate success.
*/
```

[2] Rob Murray, C++ Strategies & Tactics, Addison-Wesley, 1993, page 47.

```
int main()
{
   Matrix2x2 m1(2, -1, 1, 2); // test constructor
   cout << "m1\n" << m1 << endl;    // operator<<, the output operator

   Matrix2x2  m1Inv = m1.inverse();    // inverse
   cout << "m1.invers()\n" << m1Inv << endl;

   Matrix2x2 m1Inv_times_m1 = m1Inv*m1;
   cout << "m1 * m1.invers()\n" << m1Inv_times_m1  << endl;
   // the inverse of any 2x2 mutiplied by the 2x2 itself must give the identity 2x2
   assert(m1Inv_times_m1 == Matrix2x2(1, 0, 0, 1));

   Matrix2x2 m1_times_m1Inv = m1 * m1Inv;
   cout << "m1.invers() * m1\n" << m1_times_m1Inv << endl;
   // any 2x2 mutiplied by its inverse must give the identity 2x2
   assert(m1_times_m1Inv == Matrix2x2(1,0,0,1));

   cout << "det(m1) =   " << m1.determinant() << "\n";
   cout << "trace(m1) = " << m1.trace() << "\n\n";

   // test function object, operator()(int)
   std::vector<double > root1 = m1(1); // real = 2.0, imag = 1
   assert(std::abs(root1[0] - 2) < 1.e-6);
   assert(std::abs(root1[1] - 1) < 1.e-6);
   // implement this free function to print a given eigenvalue (see output)
   printEigenvalues(root1, 1);    // root 1: 2 +1i
   // test function object, operator()(int)
   std::vector<double > root2 ( m1(2)); // real = 2.0, imag = -1
   assert(std::abs(root2[0] - 2) < 1.e-6);
   assert(std::abs(root2[1] - (-1)) < 1.e-6);
   // implement this free function to print a given eigenvalue
   printEigenvalues(root2, 2);   // root 2: 2 -1i

   cout << "\n";

   Matrix2x2 m2 = m1 + 1;  // Mat + int, and assignment op=
   assert(m2 == Matrix2x2(3, 0, 2, 3));
   cout << "m2\n" << m2 << endl;

   m2 = 1 + m1;  // op=, int + Mat
   assert(m2 == Matrix2x2(3, 0, 2, 3));

   Matrix2x2 m3 = m2 - 1;  // Mat -int
   assert(m3 == m1);
   cout << "m3\n" << m3 << endl;

   Matrix2x2 m4 = 1 - m3; // int - Mat
   cout << "m4\n" << m4 << endl;
   assert(m4 == Matrix2x2(-1, 2, 0, -1));
```

```
    Matrix2x2 m5 =  m4 * 5 ; // Mat * int
    cout << "m5\n" << m5 << endl;
    assert(m5 == Matrix2x2(-5, 10, 0, -5));

    Matrix2x2 m6 = 10 *  m5; // int * Mat
    cout << "m6\n" << m6 << endl;
    assert(m6 == Matrix2x2(-50, 100, 0, -50));
    assert(m6 / 10 == m5);  // Mat / int
    assert(10/m6  == 10*m6.inverse());  // int / Mat, inverse
    assert(5 * m4  * 10 == m6);  // int * Mat * int == Mat

    Matrix2x2 m7 = m1++;
    cout << "m1\n" << m1 << endl;
    cout << "m7\n" << m7 << endl;
    assert(m7 == m1 - Matrix2x2(1, 1, 1, 1));

    Matrix2x2 m8 = --m1;  // --Mat
    cout << "m1\n" << m1 << endl;
    cout << "m8\n" << m8 << endl;
    assert(m8 == m1 );
    m8--;  // Mat--
    cout << "m8\n" << m8 << endl;
    assert(m1 == 1 + m8);
    assert(m1 - 1 == m8);
    assert(-m1 + 1 == -m8);
    assert(2 * m1 == m8 + m1 + 1);
    assert(m1 * m1 == m1 *(1 + m8));
    cout << "m8 is " << (m8.isSymetric() ? "" : "not") << " symmetric\n";
    Matrix2x2 m9(123, 6, 6, 4567.89);
    cout << "m9\n" << m9 << endl;
    cout << "m9 is " << (m9.isSymetric() ? "" : "not") << " symmetric\n";

    // subscripts (non-const)
    m9[0] = 3;
    m9[1] = 1;
    m9[2] = 7;
    m9[3] = 4;
    cout << "m9\n" << m9 << endl;
    assert(m9 == Matrix2x2(3, 1, 7, 4));
    cout << "det(m1) =   " << m1() << "\ntrace(m1) = " << m1.trace() << "\n\n";
    cout << "det(m9) =   " << m9() << "\ntrace(m9) = " << m9.trace() << "\n\n";
    cout << "m9 is " << (m9.isSimilar(m1) ? "" : "not") << " similar to m1\n";

    // subscripts (const version)
    const Matrix2x2 cm9{ m9 };
    cout << "cm9\n" << cm9 << endl;

    m9 += m9;
    cout << "m9\n" << m9 << endl;
    assert(m9 == 2 * Matrix2x2(3, 1, 7, 4));
```

```

    Matrix2x2 m10;
    m10 += (m9 / 2);
    cout << "m10\n" << m10 << endl;
    assert(m10 == Matrix2x2(3, 1, 7, 4));

    m10 *= 2;
    cout << "m10\n" << m10 << endl;
    assert(m10 == m9);

    m10 /= 2;
    cout << "m10\n" << m10 << endl;
    assert(m10 == m9/2);

    m10 += 10;
    cout << "m10\n" << m10 << endl;
    assert(m10 == (m9 +20) / 2);

    m10 -= 10;
    cout << "m10\n" << m10 << endl;
    assert(m10 == 0.5 * m9);
```

```
// testing transpose()
Matrix2x2 m11 = m10.transpose();
cout << "m11\n" << m11 << endl;
assert(m10 == m11.transpose());

//testing operator>>
Matrix2x2 m12;

cout << "In response to the following prompt, \n";
cout << "enter the numbers 10, 20, 30, 40, in that order\n\n";

cin >> m12;
cout << "\nm12\n" << m12 << endl;
assert(m12 == Matrix2x2(10, 20, 30, 40));

// testing unary operators + and -
Matrix2x2 m13 = -m11;
cout << "m13\n" << m13 << endl;
assert(+m11 == -m13);
```

```

    cout << "Test completed successfully!" << endl;
    return 0;
}
```

7 Output

Output of the sample test driver

```
m1
|2.00 -1.00|
|          |
|1.00  2.00|

m1.invers()
| 0.40 0.20|
|          |
|-0.20 0.40|

m1 * m1.invers()
|1.00 0.00|
|         |
|0.00 1.00|

m1.invers() * m1
|1.00 0.00|
|         |
|0.00 1.00|

det(m1) =   5
trace(m1) = 4

root 1: 2 +1i
root 2: 2 -1i

m2
|3.00 0.00|
|         |
|2.00 3.00|

m3
|2.00 -1.00|
|          |
|1.00  2.00|

m4
|-1.00  2.00|
|           |
| 0.00 -1.00|

m5
|-5.00 10.00|
|           |
| 0.00 -5.00|
```

Output of the sample test driver

```

m6
|-50.00 100.00|
|             |
|  0.00 -50.00|
m1
|3.00 0.00|
|         |
|2.00 3.00|

m7
|2.00 -1.00|
|          |
|1.00  2.00|

m1
|2.00 -1.00|
|          |
|1.00  2.00|

m8
|2.00 -1.00|
|          |
|1.00  2.00|

m8
|1.00 -2.00|
|          |
|0.00  1.00|

m8 is not symmetric
m9
|123.00    6.00|
|              |
|  6.00 4567.89|

m9 is  symmetric
m9
|3.00 1.00|
|         |
|7.00 4.00|

det(m1) =   5
trace(m1) = 4

det(m9) =   5
trace(m9) = 7

m9 is not similar to m1
```

Output of the sample test driver

```
cm9
|3.00 1.00|
|         |
|7.00 4.00|

m9
| 6.00 2.00|
|          |
|14.00 8.00|

m10
|3.00 1.00|
|         |
|7.00 4.00|

m10
| 6.00 2.00|
|          |
|14.00 8.00|

m10
|3.00 1.00|
|         |
|7.00 4.00|

m10
|13.00 11.00|
|           |
|17.00 14.00|

m10
|3.00 1.00|
|         |
|7.00 4.00|
```

```
1
2  m11
3  |3.00 7.00|
4  |         |
5  |1.00 4.00|
6
7  In response to the following prompt,
8  enter the numbers 10, 20, 30, 40, in that order
9
10 To create the following 2x2 matrix:
11 |a  b|
12 |    |
13 |c  d|
14 enter four numbers a, b, c, d, in that order:
15 10 20 30 40
16
17 m12
18 |10.00 20.00|
19 |           |
20 |30.00 40.00|
21
22 m13
23 |-3.00 -7.00|
24 |           |
25 |-1.00 -4.00|
```

Note: lines 10-14 are displayed by the input operator>> to prompt the user to enter four numbers.

Output of the sample test driver

```

Test completed successfully!
```

8 Marking scheme

60%	Program correctness:
15%	Program design, encapsulation, information hiding, code reuse, proper use of C++ concepts.
10%	No use of operator **new** and operator **delete**. No C-style coding and memory functions such as **malloc**, **alloc**, **realloc**, **free**, etc.
5%	Format, clarity, completeness of output
10%	Javadoc style documentation before introduction of every class and function, Concise documentation of nontrivial steps in code, choice of variable names, indentation and readability of program