Week 15

Remind

- 抄襲一律 0 分 （包含被抄襲者）
- 繳交期限: **12/27(Sun.) 11:59 p.m.**
- 繳交的檔案格式、名稱請符合以下規定
  - 請繳交 zip檔至 Ceiba作業區，名稱為 <student_id>.zip
  - 解壓縮後須符合格式、名稱
  - e.g. b12345678.zip
- 必須完成 Demo 才可以提早離開
- 若沒有完成 Demo 就中途早退，視同缺席
- 若當天沒有完成Demo，請以螢幕錄影解釋程式碼，並於繳交期限前將影片寄至助教信箱
  ee10042020@gmail.com

Problem - Matrix operations (11%)

Introduction

Implement the following matrix class. Note that you should use the template syntax, so that all elements in matrices can be of both type `double` and `int`.

Public

```
Matrix();
~Matrix();
Matrix(const Matrix<T> &a);
Matrix<T> operator+(const Matrix<T>&) const;
Matrix<T> operator-(const Matrix<T>&) const;
Matrix<T> operator*(const Matrix<T>&) const;
Matrix<T>& operator=(const Matrix<T>& M);
Matrix<T> operator*(const T& scalar) const ;
```

```
Matrix<T> transpose();
T determinant();
Matrix<T> &RREF();
Matrix<T> &inverse();


template<class U>
friend istream& operator>>(istream& i, Matrix<U>& M);
template<class U>
friend ostream& operator<<(ostream&, const Matrix<U>&);
template<class U>
friend Matrix<U> operator*(U, Matrix<U>&);
```

Private

```
T **matrix;
int row;
int col;
Matrix<T> &rowSwap(int row1, int row2);
Matrix<T> &rowAddTo(int row1, int row2, double scalar);
Matrix<T> &rowScale(int row, double scalar);
```

Explanation

Matrix class

```
template<class T>
class Matrix
{
public:
    Matrix();
    ~Matrix();
    Matrix(Matrix<T> &a);
    Matrix<T> operator+(const Matrix<T>&) const;
    Matrix<T> operator-(const Matrix<T>&) const;
    Matrix<T> operator*(const Matrix<T>&) const;
    Matrix<T>& operator=(const Matrix<T>& M);
    Matrix<T> operator*(const T& scaler) const ;
    Matrix<T> transpose();
    T determinant();
    Matrix<T> &RREF();
    Matrix<T> &inverse();


    template<class U>
```

```
    friend istream& operator>>(istream& i, Matrix<U>& M);
    template<class U>
    friend ostream& operator<<(ostream&, const Matrix<U>&);
    template<class U>
    friend Matrix<U> operator*(U, Matrix<U>&);

private:
    T **matrix;
    int row;
    int col;
    Matrix<T> &rowSwap(int row1, int row2);
    Matrix<T> &rowAddTo(int row1, int row2, double scalar);
    Matrix<T> &rowScale(int row, double scalar);
};
```

Constructor

Allocate memory space you need to instantiate the `Matrix` class. Set the pointer `matrix` to `NULL`.

```
Matrix();
```

Copy constructor

Allocate memory space you need to instantiate the `Matrix` class by copy constructor.

```
Matrix(Matrix<T> &A)
```

Destructor

Release all memory space you use to destruct a class `Matrix`.

```
~Matrix();
```

Matrix Addition `Matrix<T> operator+(const Matrix<T>&) const;`

`A + B` return the addition of two matrix `A` and `B`.

```
C = A + B;
```

Matrix Subtraction `Matrix<T> operator-(const Matrix<T>&) const;`

`A - B` return the difference of two matrix `A` and `B`.

```
C = A - B;
```

Matrix Multiplication `Matrix<T> operator*(const Matrix<T>&) const;`

`A * B` return the multiplication of two matrix `A` and `B`.

```
C = A * B;
```

Copier `Matrix<T>& operator=(const Matrix<T>&)`

`A = B` copy matrix `B` to matrix `A` and return matrix `A` itself .

```
A = B;
```

Scalar Multiplication `Matrix<T> operator*(const U&) const;`

`A * 3` return the multiplication of a matrix `A` and a scalar `3`.

```
C = A * 3;
```

Matrix Transpose `Matrix<T> transpose(void);`

`A.transpose()` return the transpose of a matrix `A`.

```
C = A.transpose();
```

Matrix Determinant `T determinant(void);`

`A.determinant()` return the determinant of a matrix `A`.

```
C = A.determinant();
```

There are two main ways to compute the determinant of a matrix:

1. The Leibniz method (iterative)
2. The Laplace method (recursive)
    - Hint : you can implement this function to help you calculating the determinant

      `T det(bool *col_clear, bool *row_clear, int dim)`

        - `bool *col_clear` : a boolean array records which columns have been eliminated.
        - `bool *row_clear` : a boolean array records which rows have been eliminated.
        - `int dim`: the dimension of the matrix.

Please Google these two terms for more information. Wikipedia is a nice place to start.

Reduced Row Echelon Form `Matrix<T> RREF(void);`

`A.RREF()` return the reduced row echelon form of a matrix `A`. A matrix is in row echelon form if

- all rows consisting of only zeroes are at the bottom.
- the leading coefficient (also called the pivot) of a nonzero row is always strictly to the right of the leading coefficient of the row above it.
- The leading entry in each nonzero row is a 1 (called a leading 1).
- Each column containing a leading 1 has zeros in all its other entries.

To find the RREF of a matrix, the three row operations are needed. To be precise, the three row operations consist of

- Row swapping - A row within the matrix can be switched with another row.
- Row scaling - Each element in a row can be multiplied by a non-zero constant.

- Row addition - A row can be replaced by the sum of that row and a multiple of another row.

Please implement these three as private functions of the Matrix class.

```
C = A.RREF();
```

Matrix Inverse `Matrix<T> inverse(void);`

`A.inverse()` return the inverse of a matrix `A`.

Perform Gaussian Elimination on $[A|I]$. Reduce it to $[I|A^{-1}]$, and you will get A^{-1}, which is the inverse of matrix `A`.

See more about Gaussian Elimination:
https://en.wikipedia.org/wiki/Gaussian_elimination

```
C = A.inverse();
```

Standard input overload

```
template<class T>
friend istream& operator>>(istream& i, Matrix<T>&)
```

Input a matrix `A` by using `cin >>`.

```
cin >> A;
```

Standard output overload

```
template<class T>
friend ostream& operator<<(ostream& o, const Matrix<T>&)
```

Output a matrix `A` by using `cout <<`.

```
cout << A;
```

Integer multiplication overload

```
template<class T>
friend Matrix<T> operator*(T, Matrix<U>&)
```

`3 * A` return the multiplication of a scalar `3` and a matrix `A`

```
C = 3 * A
```

Grading

(2%) Constructors, Destructor, I/O overload

(1%) Addition, Subtract

(2%) Multiplication, Scalar

(3%) Determinant

(3%) Inverse, RREF

File

Matrix.cpp