A Class for Matrices

NAME： 林洁芳

ID : 12011543

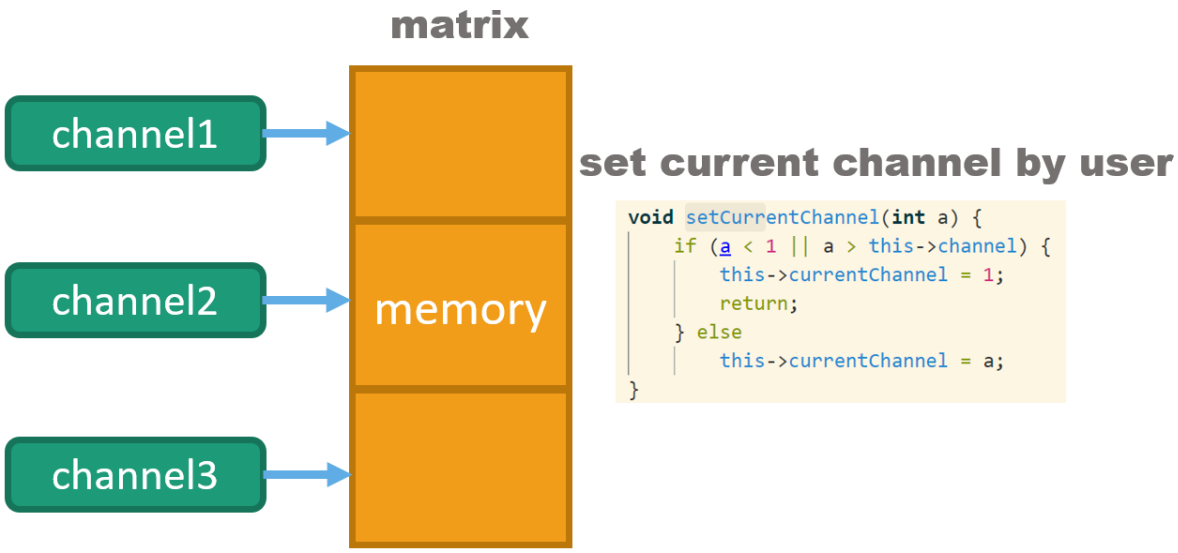
Note:

The test was conducted on VScode and Clion at the same time. Since the output of VScode was too small and arranged too closely, the display effect was not good, so most of the output of the display was from Clion.

① You can connect to each specific section through the directory.

②The default matrices in this report are two-dimensional and mainly used for image processing.

③about channel



④To avoid connection errors, place both class and function implementations in a single HPP file. (In previous attempts, a connection error occurred because the class was in an HPP file and the method was written separately in a CPP file.)

⑤In project3, the implementation of SIMD is not portable because of the uncertainty of data type and matrix size, and the architecture of ARM is different from x86. ARM also has SIMD-like methods to match its architecture. Therefore, this project adopts matrix multiplication optimized by hardware.

1 Thought & Analysis

①Need to implement a variety of data types, you can consider using template classes or learn OpenCV writing using **flag**, and finally use **template classes for implementation**.

②Shallow copy is used to strictly manage memory and avoid memory leaks. **Using int* pointer（learn from OpenCV）, analog intelligent pointer, memory maintenance.**

In addition, the following **libraries** are used for this project :

```
#include <iostream>
#include <fstream>
#include <chrono>
```

2 Code & Test Result & Description

Firstly, show all the **declaration of functions**. ***(The complete code is attached as a file.)***

```
#ifndef _MAT_H
#define _MAT_H

#include <iostream>
#include <fstream>
using namespace std;

template<typename T>
class Matrix {
private:
    int currentChannel; //the current channel
    int rank;
    int column;
    int row;
    int channel;
    int size;// row * column = size
    int step;
    int *pointTime;
    T *matrix;
public:
    int getRow() {
        return this->row;
    }

    int getCurrentChannel() {
        return this->currentChannel;
    }
    // set the current channel by this function
    void setCurrentChannel(int a) {
        if (a < 1 || a > this->channel) {
            this->currentChannel = 1;
            return;
        } else
```

```
            this->currentChannel = a;
    }

    int getSize() {
        return this->size;
    }

    T *getMatrix() {
        return this->matrix;
    }

    int *getPointTime() {
        return this->pointTime;
    }

    int getColumn() {
        return this->column;
    }

    int getStep() {
        return this->step;
    }

    int getChannel() {
        return this->channel;
    }

    int getRank() {
        return this->rank;
    }

    T getValue(int i, int j) const {
        if (i >= 0 && j >= 0 && i < this->row && j < this->column) {
            return this->matrix[(getCurrentChannel() - 1) * this->size + i *
this->step + j];
        } else return 0;
    }

    void setMatPoint(int i, int j, T num) {
        if (i >= 0 && j >= 0 && i < this->row && j < this->column) {
            this->matrix[(getCurrentChannel() - 1) * this->size + i * this->step
+ j] = num;
        } else return;
    }

    bool release() {
        this->column = 0;
        this->row = 0;
        this->step = 0;
        this->rank = 0;
        this->channel = 0;
        this->currentChannel = 0;
        if (this->matrix != nullptr) {
            delete[] this->matrix;
            this->matrix = nullptr;
        }
        return true;
    }
```

```
    ~Matrix() {
//        cout << "pointer = " << static_cast<void *>(this) << ", matrix pointer
= " << static_cast<void *>(this->matrix) << ", pointTime = " << (*this-
>pointTime) << endl;
        if (this->pointTime[0] - 1 == 0)
            release();
        else this->pointTime[0]--;
    }

    bool create(int r = 0, int c = 0, int s = 0, int ra = 0, int ch = 1) {
        release();
        this->pointTime = new int{1};
        this->column = c;
        this->row = r;
        this->step = s;
        this->rank = ra;
        this->channel = ch;
        this->currentChannel = ch;
        this->size = this->column * this->row;
        if (this->row != 0 && this->column != 0) {
            this->matrix = new T[r * c * ch];
            for (int i = 0; i < this->row * this->column * this->channel; ++i) {
                this->matrix[i] = 0;
            }
        } else this->matrix = nullptr;
        return true;
    }

    Matrix();

    Matrix(int r, int c, int ch);

    Matrix(int r, int c);

    explicit Matrix(T a);

    Matrix(int r, int c, T *a);

    Matrix(int r, int c, int ch, T *a);

    Matrix(const Matrix &);

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

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

    friend Matrix<T> operator+(const T &a, const Matrix<T> &m) {
        if (m.matrix == nullptr) {
            cerr << "PLUS FAILURE! MATRIX ERROR!" << endl;
            return Matrix<T>();
        }
        Matrix<T> res = Matrix<T>(m.row, m.column);
        for (int i = 0; i < m.row; i++) {
            for (int j = 0; j < m.column; j++) {
                res.matrix[(res.currentChannel - 1) * res.size + i * res.step +
j] =
```

```
                        m.matrix[(m.currentChannel - 1) * m.size + i * m.step +
j] + a;
            }
        }
        return res;
    }

    friend Matrix<T> operator-(const T &a, const Matrix<T> &m) {
        if (m.matrix == nullptr) {
            cerr << "SUBTRACT FAILURE! MATRIX ERROR!" << endl;
            return Matrix<T>();
        }
        Matrix<T> res = Matrix<T>(m.row, m.column);
        for (int i = 0; i < m.row; i++) {
            for (int j = 0; j < m.column; j++) {
                res.matrix[(res.currentChannel - 1) * res.size + i * res.step +
j] =
                        m.matrix[(m.currentChannel - 1) * m.size + i * m.step +
j] - a;
            }
        }
        return res;
    }

    Matrix operator+(T) const;

    Matrix operator-(T) const;

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

    friend Matrix<T> operator*(const T &a, const Matrix<T> &m) {
        if (m.matrix == nullptr) {
            cerr << "MUL FAILURE! MATRIX ERROR!" << endl;
            return Matrix<T>();
        }
        Matrix<T> res = Matrix<T>(m.row, m.column);
        for (int i = 0; i < m.row; i++) {
            for (int j = 0; j < m.column; j++) {
                res.matrix[(res.currentChannel - 1) * res.size + i * res.step +
j] =
                        m.matrix[(m.currentChannel - 1) * m.size + i * m.step +
j] * a;
            }
        }
        return res;
    }

    Matrix operator*(T) const;

    T getMin() const;

    T getMax() const;

    T getAverage() const;

    Matrix operator/(T) const;

    T operator/(const Matrix<T> &) const;
```

```
    friend T operator/(const T &a, const Matrix<T> &b) {
        if (b.getAverage() == 0 || b.matrix == nullptr) {
            cerr << "DIVISION FAILURE!" << endl;
            return 0;
        }
        return a / b.getAverage();
    }

    Matrix &operator+=(const Matrix<T> &);

    Matrix &operator-=(const Matrix<T> &);

    Matrix &operator*=(const Matrix<T> &);

    Matrix &operator/=(const Matrix<T> &);

    Matrix &operator-=(T);

    Matrix &operator/=(T);

    Matrix &operator*=(T);

    Matrix &operator+=(T);

    friend T &operator*=(T a, const Matrix<T> &b) {
        return a * b.getAverage();
    }

    friend T &operator+=(T a, const Matrix<T> &b) {
        return a + b.getAverage();
    }

    friend T &operator-=(T a, const Matrix<T> &b) {
        return a - b.getAverage();
    }

    friend T &operator/=(T &a, const Matrix<T> &b) {
        if (b.getAverage() == 0) {
            cerr << "DIVISION FAILURE!" << endl;
        } else a = a / b.getAverage();
        return a;
    }

    bool operator==(const Matrix<T> &);

    bool operator!=(const Matrix<T> &);

    explicit operator T() const {
        return this->getAverage();
    }

    void printMatrix() const;

    friend std::ostream &operator<<(std::ostream &os, const Matrix<T> &a) {
        if (a.matrix == nullptr) {
            cerr << "OUTPUT ERROR!" << endl;
            return os;
```

```
        }
        std::string str = "row = " + std::to_string(a.row) + ", column = " +
std::to_string(a.column);
        os << str << endl;
        for (int i = 0; i < a.row; ++i) {
            for (int j = 0; j < a.column; ++j) {
                string t = to_string(a.matrix[a.size * (a.currentChannel - 1) +
                                              i * a.step + j]) + " ";
                os << t;
            }
            os << endl;
        }
        return os;
    };

    friend std::istream &operator>>(std::istream &is, Matrix<T> &a) {
        for (int i = 0; i < a.row; i++) {
            for (int j = 0; j < a.column; j++) {
                is >> a.matrix[a.size * (a.currentChannel - 1) + i * a.step +
j];
            }
        }
        return is;
    }

    Matrix getUnit();

    T getTrace() const;

    Matrix getTranspose() const;

    T getRowAve(int i) const;

    T getColumnAve(int i) const;

    Matrix removeRow(int) const;

    Matrix removeColumn(int) const;

    void changeSize(int i, int j);

    void outputToFile(std::ofstream &);

    void readFromFile(const string &, int);

    Matrix LU(Matrix<T> &) const;

    void swapRow(int, int);

    void scaleRow(int, T);

    T getDet() const;

    Matrix inverse();

    Matrix getRankLadder();

    Matrix getAdjoin() const {
```

```
        return this->inverse() * this->getDet();
    }

    Matrix &operator=(const Matrix<T> &);

    Matrix getSubMat(int i, int j, int r, int c);

private:
    //be used inside the class
    void exchangeRow(T *, int, int, int, int);

    void mulRow(T *, int, T, int, int);

    void unitization(T *, int, int, int);

    void addRow(T *, int, int, T, int, int);

};

#endif
```

Some short functions and all of the friend functions are defined inside the class.

Friend functions declared in a template class, defined outside the template class are prone to link failures. A friend function, like a member function of a template class, has direct access to the class's member variables and methods.

2.1 friend function

2.1.1 + & - & * & /

To avoid errors in the following situations :

```
// In 100, for example :
Matrix<double> ap = 100 + a;
Matrix<double> as = 100 - a;
Matrix<double> am = 100 * a;
double ai = 100 / a;
```

Because the symbol is preceded by the primitive data type (not the Matrix type), the friend function is overwritten.

The operation rules of addition, subtraction and multiplication are very intuitive : each element in the matrix is respectively added, subtracted and multiplied.

```
friend Matrix<T> operator+(const T &a, const Matrix<T> &m) {
    if (m.matrix == nullptr) {
        cerr << "PLUS FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(m.row, m.column);
    for (int i = 0; i < m.row; i++) {
        for (int j = 0; j < m.column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    m.matrix[(m.currentChannel - 1) * m.size + i * m.step + j] +
a;
```

```
        }
    }
    return res;
}

friend Matrix<T> operator-(const T &a, const Matrix<T> &m) {
    if (m.matrix == nullptr) {
        cerr << "SUBTRACT FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(m.row, m.column);
    for (int i = 0; i < m.row; i++) {
        for (int j = 0; j < m.column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    m.matrix[(m.currentChannel - 1) * m.size + i * m.step + j] -
a;
        }
    }
    return res;
}

friend Matrix<T> operator*(const T &a, const Matrix<T> &m) {
        if (m.matrix == nullptr) {
            cerr << "MUL FAILURE! MATRIX ERROR!" << endl;
            return Matrix<T>();
        }
        Matrix<T> res = Matrix<T>(m.row, m.column);
        for (int i = 0; i < m.row; i++) {
            for (int j = 0; j < m.column; j++) {
                res.matrix[(res.currentChannel - 1) * res.size + i * res.step +
j] =
                        m.matrix[(m.currentChannel - 1) * m.size + i * m.step +
j] * a;
            }
        }
        return res;
    }
```

For division operations, since division is not commutative, division is defined in the following form :

```
100 / a matrix = 100 / the average of all the elements of the matrix, and return
a number;
```

So rewrite the method as follows :

```
friend T operator/(const T &a, const Matrix<T> &b) {
        if (b.getAverage() == 0 || b.matrix == nullptr) {
            cerr << "DIVISION FAILURE!" << endl;
            return 0;
        }
        return a / b.getAverage();
}
```

The code execution result is as follows :

①a sample matrices of size 6 * 6：

```
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
```

②+ & - & * & / execution result：

```
Matrix<long> ap = 100 + a;
Matrix<long> as = 100 - a;
Matrix<long> am = 100 * a;
long ai = 100 / a;
cout << "Matrix<long> ap = 100 + a : " << endl ;
cout << ap;
cout << "Matrix<long> as = 100 - a : " << endl ;
cout << as;
cout << "Matrix<long> am = 100 * a : " << endl ;
cout << am;
cout << "long ai = 100 / a : " << endl ;
cout << ai;
```

```
Matrix<long> ap = 100 + a :
row = 6, column = 6
101 102 103 104 105 106
107 108 109 110 111 112
102 103 104 105 106 107
104 105 106 107 108 109
106 107 108 109 110 111
101 103 105 107 109 111
Matrix<long> as = 100 - a :
row = 6, column = 6
-99 -98 -97 -96 -95 -94
-93 -92 -91 -90 -89 -88
-98 -97 -96 -95 -94 -93
-96 -95 -94 -93 -92 -91
-94 -93 -92 -91 -90 -89
-99 -97 -95 -93 -91 -89
Matrix<long> am = 100 * a :
row = 6, column = 6
100 200 300 400 500 600
700 800 900 1000 1100 1200
200 300 400 500 600 700
400 500 600 700 800 900
600 700 800 900 1000 1100
100 300 500 700 900 1100
long ai = 100 / a :
16
Process finished with exit code 0
```

2.1.2 << & >>

Overloading input and output streams mainly facilitates matrix input and output. The code is shown below :

```
friend std::ostream &operator<<(std::ostream &os, const Matrix<T> &a) {
        if (a.matrix == nullptr) {
            cerr << "OUTPUT ERROR!" << endl;
            return os;
        }
        std::string str = "row = " + std::to_string(a.row) + ", column = " +
std::to_string(a.column);
        os << str << endl;
        for (int i = 0; i < a.row; ++i) {
            for (int j = 0; j < a.column; ++j) {
                string t = to_string(a.matrix[a.size * (a.currentChannel - 1) +
                                              i * a.step + j]) + " ";
                os << t;
            }
            os << endl;
        }
        return os;
}
```

```
friend std::istream &operator>>(std::istream &is, Matrix<T> &a) {
        for (int i = 0; i < a.row; i++) {
            for (int j = 0; j < a.column; j++) {
                is >> a.matrix[a.size * (a.currentChannel - 1) + i * a.step +
j];
            }
        }
        return is;
}
```

The operator ">>" can read the data one size (row * column) at a time. For example, I want to read in a 2 by 6 matrix, and it has three channels, then show it as follows :

```
// the test code as follows(the test of << is also contained):

Matrix<long> a = Matrix<long>(2, 6, 3);
a.setCurrentChannel(2);
cout << "channel 2 :" << endl;
cin >> a;
a.setCurrentChannel(1);
cout << "channel 1 :" << endl;
cin >> a;
a.setCurrentChannel(3);
cout << "channel 3 :" << endl;
cin >> a;
cout << "a : " << endl;
for (int i = 1; i <= a.getChannel(); ++i) {
    a.setCurrentChannel(i);
    cout << a;
}
```

the result shows here,

```
channel 2 :
1 2 3 4 5 6
7 8 9 10 11 12
channel 1 :
2 3 4 5 6 7
4 5 6 7 8 9
channel 3 :
6 7 8 9 10 11
1 3 5 7 9 11
a :
row = 2, column = 6
2 3 4 5 6 7
4 5 6 7 8 9
row = 2, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
row = 2, column = 6
6 7 8 9 10 11
1 3 5 7 9 11

Process finished with exit code 0
```

The storage details are explained in the constructor functions that follow.

2.1.3 += & -= & *= & /=

"+=", "-=", "*=" and "/=", the friend function of the operator above mainly deals with the calculation of operands to the left and matrices to the right of the operator.

The calculation of subtraction, addition, multiplication and division of a matrix by a number is inherently wrong, but in order to avoid errors in the program, we can only consider a custom set of rules, that is, the operation of the matrix into the operation of the matrix's current channel mean. The code is shown below :

```
friend T &operator*=(T &a, const Matrix<T> &b) {
    a = a * b.getAverage();
    return a;
}
friend T &operator+=(T &a, const Matrix<T> &b) {
    a = a + b.getAverage();
    return a;
}
friend T &operator-=(T &a, const Matrix<T> &b) {
    a = a - b.getAverage();
    return a;
}

//It is necessary to verify the dividend legally for division, that is, if the
dividend is zero, the calculation will be terminated to avoid program errors.
friend T &operator/=(T &a, const Matrix<T> &b) {
    if (b.getAverage() == 0) {
        cerr << "DIVISION FAILURE!" << endl;
    } else a = a / b.getAverage();
    return a;
```

```
}
```

The example of test code as follows :

```
Matrix<long> a = Matrix<long>(6, 6);
a.readFromFile(fileA, 1);
long tem = 66;
cout << "tem = " << tem << endl;
tem *= a;
//tem += a;
//tem -= a;
//tem /= a;
cout << "66 *= a : " << endl;
cout << a;
/*
the test data:
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
*/
```

and all the test result shows here :

```
tem = 66          tem = 66        tem = 66        tem = 66
66 /= a : 11      66 += a : 72    66 -= a : 60    66 *= a : 396
```

2.2 constructor and destructor

The following seven constructors were designed to overwrite the system's default no-argument constructor in order to better cater to various requirements.

(In addition, the memory management test code and result analysis are shown in "Difficulty and Solution".)

```
//no parameter constructor
Matrix();

//constructor that specify rows and columns and channels
Matrix(int r, int c, int ch);

//constructor that specify rows and columns
Matrix(int r, int c);

//explicitly converts the basic data types (int, double, and so on) to a one-by-one matrix and assigns the elements of the matrix to that number.
explicit Matrix(T a);

/*learn how to initialize one of OpenCV's constructors by specifying the rows and columns of a matrix and passing in an array.
(emphasis: shallow copy, i.e. the matrix in the matrix class has the same address as the array!)
*/
Matrix(int r, int c, T *a);
```

```
/*learn how to initialize one of OpenCV's constructors by specifying the rows,
columns and channels of a matrix and passing in an array.
(emphasis: shallow copy, i.e. the matrix in the matrix class has the same address
as the array!)
*/
Matrix(int r, int c, int ch, T *a);

//overwrite the copy constructor
Matrix(const Matrix &);

//overwrite the destructor
~Matrix();
```

The constructor is implemented and explained as follows :

First, show two functions used by the two constructors, **release()** and **create(int r = 0, int c = 0, int s = 0, int ra = 0, int ch = 1)**.

```
bool release() {
    this->column = 0;
    this->row = 0;
    this->step = 0;
    this->rank = 0;
    this->channel = 0;
    this->currentChannel = 0;
    if (this->matrix != nullptr) {
        delete[] this->matrix;
        this->matrix = nullptr;
    }
    return true;
}

bool create(int r = 0, int c = 0, int s = 0, int ra = 0, int ch = 1) {
    release();
    this->pointTime = new int{1};
    this->column = c;
    this->row = r;
    this->step = s;
    this->rank = ra;
    this->channel = ch;
    this->currentChannel = ch;
    this->size = this->column * this->row;
    if (this->row != 0 && this->column != 0) {
        this->matrix = new T[r * c * ch];
        for (int i = 0; i < this->row * this->column * this->channel; ++i) {
            this->matrix[i] = 0;
        }
    } else this->matrix = nullptr;
    return true;
}
```

2.2.1 Matrix();

Overwrite the system's built-in constructor, the main purpose is to initialize some member variables.

The code shows as follows :

```
template<typename T>
Matrix<T>::Matrix() {
    cout << "Matrix()" << endl;
    this->matrix = nullptr;
    create();
}
```

2.2.2 Matrix(int r, int c, int ch);

This constructor can request a matrix of a specified size from the system and differentiate between different channels. The code shows as follows :

```
template<typename T>
Matrix<T>::Matrix(int r, int c, int ch) {
    cout << "Matrix(int, int, int)" << endl;
    this->matrix = nullptr;
    create(r, c, c, 0, ch);
    cout << *(this->pointTime) << endl;
    cout << this->pointTime << endl;
}
```

2.2.3 Matrix(int r, int c);

This constructor can request a matrix of a specified size from the system to facilitate subsequent data readings. The code shows as follows :

```
template<typename T>
Matrix<T>::Matrix(int r, int c) {
    cout << "Matrix(int, int)" << endl;
    this->matrix = nullptr;
    create(r, c, c);
//    cout << "the address is " << this->pointTime << ", the num of point is " <<
this->pointTime[0] << endl;
}
```

2.2.4 explicit Matrix(T a);

This constructor casts a primitive data type to a matrix-class type and generates a one-by-one, one-element matrix that is initialized to be equal to that number. The code shows as follows :

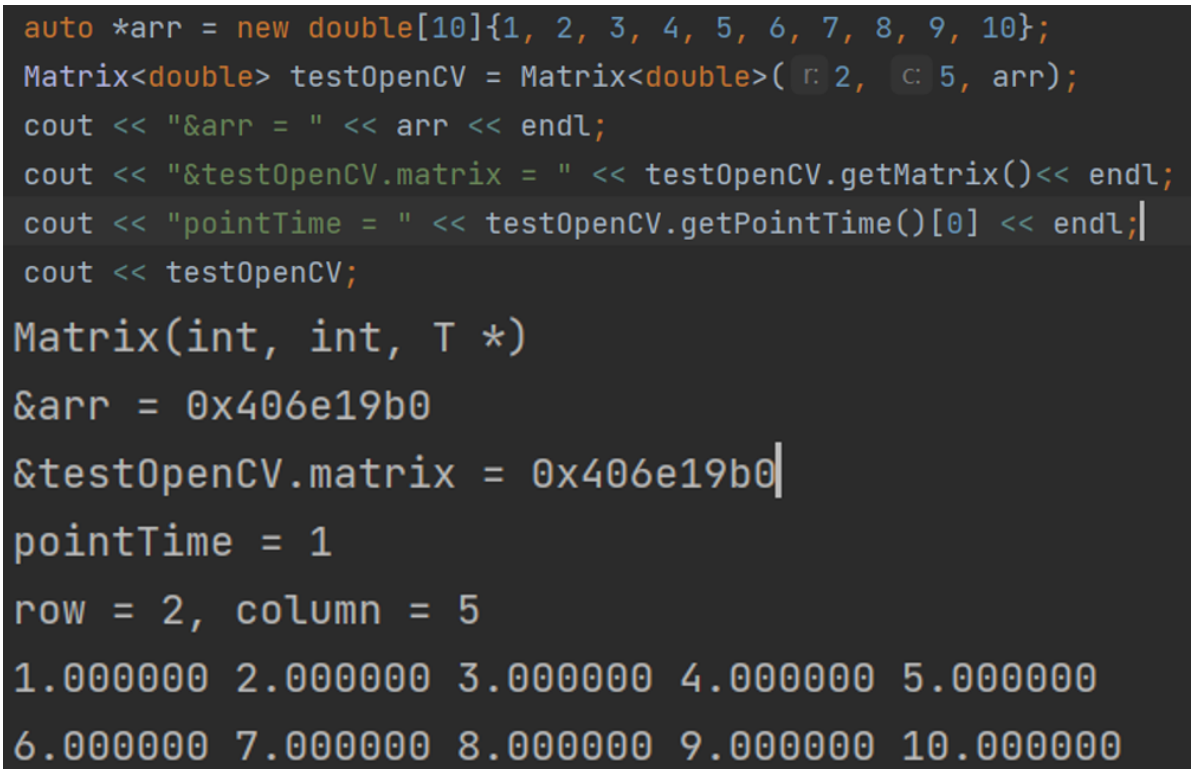
```
template<typename T>
Matrix<T>::Matrix(T a) {
    cout << "Matrix(T)" << endl;
    this->matrix = nullptr;
    create(1, 1, 1);
    this->matrix[0] = a;
}
```

2.2.5 Matrix(int r, int c, T *a);

This constructor initializes the row and row information of the matrix and makes a shallow copy of the array by pointing the matrix array pointer in the matrix class to the passed array pointer. The code shows as follows :

```
template<typename T>
Matrix<T>::Matrix(int r, int c, T *a) {
    cout << "Matrix(int, int, T *)" << endl;
    this->matrix = nullptr;
    create(r, c, c);
    delete[] this->matrix;
    this->matrix = a;
//    cout << this->matrix << endl;
}
```

The test code and results are as follows :



2.2.6 Matrix(int r, int c, int ch, T *a);

Matrix(int r, int c, T *a); , on the basis of this constructor, the matrix number component channels. The code shows as follows :

```
template<typename T>
Matrix<T>::Matrix(int r, int c, int ch, T *a) {
    cout << "Matrix(int, int, int, T* &)" << endl;
    this->matrix = nullptr;
    create(r, c, c, 0, ch);
    delete[] this->matrix;
    this->matrix = a;
}
```

2.2.7 Matrix(const Matrix &);

Rewrite the system's own copy function, the main purpose is one, can avoid memory leaks, two can maintain the pointer counter. The code shows as follows :

```
template<typename T>
Matrix<T>::Matrix(const Matrix &a) {
    cout << "Matrix(const Matrix &)" << endl;
    this->matrix = nullptr;
    create(a.row, a.column, a.step, a.rank, a.channel);
    delete[] this->matrix;
    this->matrix = a.matrix;
    this->pointTime = a.pointTime;
    this->pointTime[0]++;
//    cout << "the address is " << this->pointTime << ", the num of point is " <<
this->pointTime[0] << endl;
}
```

2.2.8 ~Matrix();

Rewrite the destructor of the system to avoid memory problems caused by premature memory release. The code shows as follows :

```
~Matrix() {
//cout << "pointer = " << static_cast<void *>(this) << ", matrix pointer = " <<
static_cast<void *>(this->matrix) << ", pointTime = " << (*this->pointTime) <<
endl;
    if (this->pointTime[0] - 1 == 0)
        release();
    else this->pointTime[0]--;
}
```

2.3 overloaded operator

In the function implementation, the function should be as robust as possible, that is, the error type of null pointer can be avoided as far as possible. If a condition is encountered that the calculation cannot continue, an error message is returned to the command line, terminating the function execution.

The sample matrices is as follows :

a

```
7.8 44.4 2.3 87.7 80.1 17.9 23.1 5.3
60.9 4.3 35.9 90.6 68.2 72.2 79.4 2.7
83.5 66.3 41.7 21.5 76.7 50.1 59.7 43.1
33.4 63 90.2 34.6 7.1 77.8 34.5 42.4
47 31 21.2 73.7 18 8.6 44.9 59.4
82.1 56.9 0 1.9 86.4 27.9 37.6 16.8
93.7 84.8 30.3 34.8 60.7 44.3 92.6 55.2
70.1 35.6 43.1 6.5 39.7 58.6 9.8 95
```

b

```
31.9 47.4 13.5 0.6 94.4 61.4 29.4 74.8
21.2 50.1 88.3 48.1 22.6 57.2 26 30.5
16.9 54.1 46.9 21.3 34.7 49.4 26.3 34.9
20 80.1 70 9.6 4.3 0.3 93.7 21.6
6 40.6 81.8 57 41.9 80.2 93.2 45.6
79.2 4 28.2 79.1 75.2 86.5 85.7 35.7
25 25.9 31.9 80.5 43.6 58.1 23.7 29
13.6 27.2 11.7 72.5 78.5 98.4 97.2 10.2
```

2.3.1 + & - & * & /

2.3.1.1 between matrices

Overload the addition, subtraction, multiplication and division operators so that "+ \ - \ * \ /" operations can be performed directly between matrix types.

In particular, the division between matrices is defined as the division operation between the average of all elements of the current channel of two matrices, and the return value is a basic data type that depends on the data type of the **data** inside the matrix class.

The code shows as follows :

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

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

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

T operator/(const Matrix<T> &) const;

//defined
template<typename T>
Matrix<T> Matrix<T>::operator+(const Matrix<T> &a) const {
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "PIUS FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    if (a.column != this->column || this->row != a.row) {
        cerr << "PLUS FAILURE!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(a.row, a.column);
    for (int i = 0; i < a.row; i++) {
        for (int j = 0; j < a.column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    a.matrix[(a.currentChannel - 1) * a.size + i * a.step + j] +
                    this->matrix[(this->currentChannel - 1) *
                                 this->size + i * a.step + j];
        }
    }
    return res;
}

template<typename T>
Matrix<T> Matrix<T>::operator-(const Matrix<T> &a) const {
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "SUBTRACT FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    if (a.column != this->column || this->row != a.row) {
        cerr << "SUBTRACT FAILURE! ROW OR COLUMN NOT MATCH!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(a.row, a.column);
    for (int i = 0; i < a.row; i++) {
        for (int j = 0; j < a.column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    a.matrix[(a.currentChannel - 1) * a.size + i * a.step + j] -
                    this->matrix[(this->currentChannel - 1) *
                                 this->size + i * a.step + j];
        }
    }
```

```
    return res;
}

template<typename T>
Matrix<T> Matrix<T>::operator*(const Matrix<T> &a) const {
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "MUL FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    if (this->column != a.row) {
        cerr << "MUL FAILURE! NOT MATCH!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(this->row, a.column);
    {
        T c;
#pragma omp parallel for
        for (int i = 0; i < this->row; i++) {
            for (int k = 0; k < this->column; k++) {
                c = this->matrix[(this->currentChannel - 1) *
                                 this->size + i * this->step + k];
                for (int j = 0; j < a.column; j++) {
                    res.matrix[(res.currentChannel - 1) * res.size + i *
res.step + j] +=
                            c * a.matrix[(a.currentChannel - 1) * a.size + k *
a.step + j];
                }
            }
        }
    }
    return res;
}

template<typename T>
T Matrix<T>::operator/(const Matrix<T> &a) const {
    if (a.getAverage() == 0 || a.matrix == nullptr || this->matrix == nullptr) {
        cerr << "DIVISION FAILURE!" << endl;
        return 0;
    }
    return this->getAverage() / a.getAverage();
}
```

The results are shown below :

```
Matrix<double> aAb = a + b;
Matrix<double> aSb = a - b;
Matrix<double> aMb = a * b;
double aCb = a / b;
aAb.outputToFile(output);
aSb.outputToFile(output);
aMb.outputToFile(output);
output << "a / b = " << std::to_string(aCb) << endl;
```

```
row = 8, column = 8
39.700000 91.800000 15.800000 88.300000 174.500000 79.300000 52.500000 80.100000
82.100000 54.400000 124.200000 138.700000 90.800000 129.400000 105.400000 33.200000
100.400000 120.400000 88.600000 42.800000 111.400000 99.500000 86.000000 78.000000
53.400000 143.100000 160.200000 44.200000 11.400000 78.100000 128.200000 64.000000
53.000000 71.600000 103.000000 130.700000 59.900000 88.800000 138.100000 105.000000
161.300000 60.900000 28.200000 81.000000 161.600000 114.400000 123.300000 52.500000
118.700000 110.700000 62.200000 115.300000 104.300000 102.400000 116.300000 84.200000
83.700000 62.800000 54.800000 79.000000 118.200000 157.000000 107.000000 105.200000

row = 8, column = 8
24.100000 3.000000 11.200000 -87.100000 14.300000 43.500000 6.300000 69.500000
-39.700000 45.800000 52.400000 -42.500000 -45.600000 -15.000000 -53.400000 27.800000
-66.600000 -12.200000 5.200000 -0.200000 -42.000000 -0.700000 -33.400000 -8.200000
-13.400000 17.100000 -20.200000 -25.000000 -2.800000 -77.500000 59.200000 -20.800000
-41.000000 9.600000 60.600000 -16.700000 23.900000 71.600000 48.300000 -13.800000
-2.900000 -52.900000 28.200000 77.200000 -11.200000 58.600000 48.100000 18.900000
-68.700000 -58.900000 1.600000 45.700000 -17.100000 13.800000 -68.900000 -26.200000
-56.500000 -8.400000 -31.400000 66.000000 38.800000 39.800000 87.400000 -84.800000
```

```
row = 8, column = 8
5530.830000 13809.470000 18128.550000 11256.620000 8322.160000 12994.530000 19723.680000 8927.780000
12601.740000 17488.960000 19406.800000 18063.670000 19442.260000 22379.620000 26023.650000 15913.940000
11710.720000 17290.620000 20537.850000 20599.150000 23887.740000 29180.290000 24336.180000 17644.690000
12260.940000 15037.030000 17037.550000 17713.690000 18836.130000 23596.350000 20502.350000 12849.360000
6708.240000 14374.980000 13367.290000 12305.590000 14211.410000 16369.540000 18903.480000 9828.700000
7759.830000 11944.660000 15515.920000 14180.880000 17720.750000 21476.500000 17038.760000 14115.200000
12933.340000 19657.970000 22424.160000 23534.900000 26208.060000 31622.830000 26031.040000 19002.180000
10265.620000 12642.700000 12890.320000 17309.410000 22900.450000 26641.770000 22917.440000 13129.410000

a / b = 1.016460
```

2.3.1.2 matrix versus operands

The calculation of the operands by the matrix itself is wrong, but here we define that each element of the current channel of the matrix performs the operation on the operands separately, resulting in a new matrix. The code shows as follows :

```
//declaration
Matrix operator/(T) const;

Matrix operator+(T) const;

Matrix operator-(T) const;

Matrix operator*(T) const;

//defined
template<typename T>
Matrix<T> Matrix<T>::operator+(T a) const {
    if (this->matrix == nullptr) {
        cerr << "PLUS FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(this->row, this->column);
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    a + this->matrix[(this->currentChannel - 1) *
                                     this->size + i * a.step + j];
        }
    }
    return res;
```

```
}

template<typename T>
Matrix<T> Matrix<T>::operator-(T a) const {
    if (this->matrix == nullptr) {
        cerr << "SUBTRACT FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(this->row, this->column);
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    this->matrix[(this->currentChannel - 1) *
                                 this->size + i * a.step + j] - a;
        }
    }
    return res;
}

template<typename T>
Matrix<T> Matrix<T>::operator*(T a) const {
    if (this->matrix == nullptr) {
        cerr << "MUL FAILURE! MATRIX ERROR!" << endl;
        return Matrix<T>();
    }
    Matrix<T> res = Matrix<T>(this->row, this->column);
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    this->matrix[(this->currentChannel - 1) *
                                 this->size + i * a.step + j] * a;
        }
    }
    return res;
}

template<typename T>
Matrix<T> Matrix<T>::operator/(T a) const {
    if (a == 0 || this->matrix == nullptr) {
        cerr << "DIVISION FAILURE!" << endl;
        return Matrix<T>();;
    }
    Matrix<T> res = Matrix<T>(this->row, this->column);
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            res.matrix[(res.currentChannel - 1) * res.size + i * res.step + j] =
                    this->matrix[(this->currentChannel - 1) * this->size + i *
this->step + j] / a;
        }
    }
    return res;
}
```

The results are shown below :

```
row = 8, column = 8
107.800000 144.400000 102.300000 187.700000 180.100000 117.900000 123.100000 105.300000
160.900000 104.300000 135.900000 190.4                                              00
183.500000 166.300000 141.700000 121.                                               00
133.400000 163.000000 190.200000 134.600000 107.100000 177.800000 134.500000 142.400000
147.000000 131.000000 121.200000 173.700000 118.000000 108.600000 144.900000 159.400000
182.100000 156.900000 100.000000 101.900000 186.400000 127.900000 137.600000 116.800000
193.700000 184.800000 130.300000 134.800000 160.700000 144.300000 192.600000 155.200000
170.100000 135.600000 143.100000 106.500000 139.700000 158.600000 109.800000 195.000000

row = 8, column = 8
-92.200000 -55.600000 -97.700000 -12.                                               00
-39.100000 -95.700000 -64.100000 -9.4                                               0
-16.500000 -33.700000 -58.300000 -78.                                               00
-66.600000 -37.000000 -9.800000 -65.400000 -92.900000 -22.200000 -65.500000 -57.600000
-53.000000 -69.000000 -78.800000 -26.300000 -82.000000 -91.400000 -55.100000 -40.600000
-17.900000 -43.100000 -100.000000 -98.100000 -13.600000 -72.100000 -62.400000 -83.200000
-6.300000 -15.200000 -69.700000 -65.200000 -39.300000 -55.700000 -7.400000 -44.800000
-29.900000 -64.400000 -56.900000 -93.500000 -60.300000 -41.400000 -90.200000 -5.000000
```

```
Matrix<double> a = Matrix<double>(8, 8);
Matrix<double> pa = a + 100;
```

```
Matrix<double> a = Matrix<double>(8, 8);
Matrix<double> sa = a - 100;
```

```
row = 8, column = 8
780.000000 4440.000000 230.000000 8770.000000 8010.000000 1790.000000 2310.000000 530.000000
6090.000000 430.000000 3590.000000 90                                               70.000000
8350.000000 6630.000000 4170.000000 2                                               4310.000000
3340.000000 6300.000000 9020.000000 3460.000000 710.000000 7780.000000 3450.000000 4240.000000
4700.000000 3100.000000 2120.000000 7370.000000 1800.000000 860.000000 4490.000000 5940.000000
8210.000000 5690.000000 0.000000 190.000000 8640.000000 2790.000000 3760.000000 1680.000000
9370.000000 8480.000000 3030.000000 3480.000000 6070.000000 4430.000000 9260.000000 5520.000000
7010.000000 3560.000000 4310.000000 650.000000 3970.000000 5860.000000 980.000000 9500.000000

row = 8, column = 8
0.078000 0.444000 0.023000 0.877000 0.801000 0.179000 0.231000 0.053000
0.609000 0.043000 0.359000 0.906000 0
0.835000 0.663000 0.417000 0.215000 0
0.334000 0.630000 0.902000 0.346000 0.071000 0.778000 0.345000 0.424000
0.470000 0.310000 0.212000 0.737000 0.180000 0.086000 0.449000 0.594000
0.821000 0.569000 0.000000 0.019000 0.864000 0.279000 0.376000 0.168000
0.937000 0.848000 0.303000 0.348000 0.607000 0.443000 0.926000 0.552000
0.701000 0.356000 0.431000 0.065000 0.397000 0.586000 0.098000 0.950000
```

```
Matrix<double> a = Matrix<double>(8, 8);
Matrix<double> ma = a * 100;
```

```
Matrix<double> a = Matrix<double>(8, 8);
Matrix<double> ia = a / 100;
```

2.3.2 += & -= & *= & /=

2.3.2.1 between matrices

The result of performing operations between matrices directly modifies the original matrix, the matrix to the left of the operator and return ***this**.

In particular, for "/=", in order to maintain the previous definition of division between matrices, that is, the division between matrices is the division of the current channel average of the two matrices, so when implementing "/=" overload, the return value needs to do some processing, that is, the original matrix, the matrix on the left of the operator free memory, and the pointer points to a newly generated matrix whose size is 1 * 1 and element values are calculated.

In addition, for "*=", if the two square matrices are not multiplied, then the size of the original matrix is likely to change. Then, if the change of the array size is within the allowed scope of the original space, the data will be updated to the original matrix, and the redundant position will be zeroed; otherwise, the calculation will be terminated to avoid memory leakage.

```
//declaration
```

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

Matrix &operator-=(const Matrix<T> &);

Matrix &operator*=(const Matrix<T> &);

Matrix &operator/=(const Matrix<T> &);

//defined
template<typename T>
Matrix<T> &Matrix<T>::operator/=(const Matrix<T> &a) {
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "DIVIDE FAILURE!" << endl;
        return *this;
    }
    if (a.getAverage() == 0) {
        cerr << "DIVIDE FAILURE!" << endl;
        return *this;
    } else {
        T ave = this->getAverage();
        this->release();
        *this = Matrix<T>(ave / a.getAverage());
        return *this;
    }
}

template<typename T>
Matrix<T> &Matrix<T>::operator-=(const Matrix<T> &a) {
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "SUBTRACT FAILURE! MATRIX ERROR!" << endl;
        return *this;
    }
    if (a.column != this->column || this->row != a.row) {
        cerr << "SUBTRACT FAILURE! NOT MATCH!" << endl;
        return *this;
    }
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            this->matrix[(this->currentChannel - 1) *
                         this->size + i * this->step + j] -=
a.matrix[(a.currentChannel - 1) * a.size + i * a.step + j];
        }
    }
    return *this;
}

template<typename T>
Matrix<T> &Matrix<T>::operator+=(const Matrix<T> &a) {
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "PLUS FAILURE! MATRIX ERROR!" << endl;
        return *this;
    }
    if (a.column != this->column || this->row != a.row) {
        cerr << "PLUS FAILURE! NOT MATCH!" << endl;
        return *this;
    }
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
```

```
            this->matrix[(this->currentChannel - 1) *
                         this->size + i * this->step + j] +=
a.matrix[(a.currentChannel - 1) * a.size + i * a.step + j];
        }
    }
    return *this;
}

template<typename T>
Matrix<T> &Matrix<T>::operator*=(const Matrix<T> &a) {
    if (this->row * a.column > this->size) {
        cerr << "MUL FAILURE! MEMORY LEAK!" << endl;
        return *this;
    }
    if (this->matrix == nullptr || a.matrix == nullptr) {
        cerr << "MUL FAILURE! MATRIX ERROR!" << endl;
        return *this;
    }
    if (this->column != a.row || this->column != a.column) {
        cerr << "MUL FAILURE! NOT MATCH!" << endl;
        return *this;
    }
    Matrix<T> res = Matrix<T>(this->row, a.column);
    {
        T c;
#pragma omp parallel for
        for (int i = 0; i < this->row; i++) {
            for (int k = 0; k < this->column; k++) {
                c = this->matrix[(this->currentChannel - 1) * this->size + i *
this->step + k];
                for (int j = 0; j < a.column; j++) {
                    res.matrix[(res.currentChannel - 1) * res.size + i *
res.step + j] +=
                            c * a.matrix[(a.currentChannel - 1) * a.size + k *
a.step + j];
                }
            }
        }
    }
    for (int i = 0; i < res.size; ++i) {
        if (i < this->size)
            this->matrix[(this->currentChannel - 1) * this->size + i] =
res.matrix[(res.currentChannel - 1) * res.size + i];
        else this->matrix[(res.currentChannel - 1) * res.size + i] = 0;
    }
    return *this;
}
```

The results are shown below :

```
Matrix<long> a = Matrix<long>( r: 8,  c: 8);
a.readFromFile( in: fileA,  type: 1);
Matrix<long> b = Matrix<long>( r: 8,  c: 8);
b.readFromFile( in: fileB,  type: 1);
cout << "a : " << endl;
cout << a;
a += b;
cout << "a += b, a : " <<endl;
cout << a;
```

```
Matrix<long> a = Matrix<long>( r: 8,  c: 8);
a.readFromFile( in: fileA,  type: 1);
Matrix<long> b = Matrix<long>( r: 8,  c: 8);
b.readFromFile( in: fileB,  type: 1);
cout << "a : " << endl;
cout << a;
a -= b;
cout << "a -= b, a : " <<endl;
cout << a;
```

```
Matrix<long> a = Matrix<long>( r: 8,  c: 8);
a.readFromFile( in: fileA,  type: 1);
Matrix<long> b = Matrix<long>( r: 8,  c: 4);
b.readFromFile( in: fileB,  type: 1);
cout << "a : " << endl;
cout << a;
a *= b;
cout << "a *= b, a : " <<endl;
cout << a;
```

```
Matrix<long> a = Matrix<long>( r: 8,  c: 8);
a.readFromFile( in: fileA,  type: 1);
Matrix<long> b = Matrix<long>( r: 8,  c: 8);
b.readFromFile( in: fileB,  type: 1);
cout << "a : " << endl;
cout << a;
a /= b;
cout << "a /= b, a : " <<endl;
cout << a;
```

```
a :
row = 8, column = 8
7 44 2 87 80 17 23 5
60 4 35 90 68 72 79 2
83 66 41 21 76 50 59 43
33 63 90 34 7 77 34 42
47 31 21 73 18 8 44 59
82 56 0 1 86 27 37 16
93 84 30 34 60 44 92 55
70 35 43 6 39 58 9 95
a += b, a :
row = 8, column = 8
38 91 15 87 174 78 52 79
81 54 123 138 90 129 105 32
99 120 87 42 110 99 85 77
53 143 160 43 11 77 127 63
53 71 102 130 59 88 137 104
161 60 28 80 161 113 122 51
118 109 61 114 103 102 115 84
83 62 54 78 117 156 106 105
```

```
a :
row = 8, column = 8
7 44 2 87 80 17 23 5
60 4 35 90 68 72 79 2
83 66 41 21 76 50 59 43
33 63 90 34 7 77 34 42
47 31 21 73 18 8 44 59
82 56 0 1 86 27 37 16
93 84 30 34 60 44 92 55
70 35 43 6 39 58 9 95
a -= b, a :
row = 8, column = 8
-24 -3 -11 87 -14 -44 -6 -69
39 -46 -53 42 46 15 53 -28
67 12 -5 0 42 1 33 9
13 -17 20 25 3 77 -59 21
41 -9 -60 16 -23 -72 -49 14
3 52 -28 -78 11 -59 -48 -19
68 59 -1 -46 17 -14 69 26
57 8 32 -66 -39 -40 -88 85
```

```
a :
row = 8, column = 8
7 44 2 87 80 17 23 5
60 4 35 90 68 72 79 2
83 66 41 21 76 50 59 43
33 63 90 34 7 77 34 42
47 31 21 73 18 8 44 59
82 56 0 1 86 27 37 16
93 84 30 34 60 44 92 55
70 35 43 6 39 58 9 95
Matrix(int, int)
a *= b, a :
row = 8, column = 8
8647 15065 10002 8532 10075 23464 17032 9305
14368 22448 24443 12332 11563 17199 18669 11297
8094 10835 14523 8980 10926 16254 11452 7567
9435 18219 18800 7339 9651 13585 18632 9701
47 31 21 73 18 8 44 59
82 56 0 1 86 27 37 16
93 84 30 34 60 44 92 55
70 35 43 6 39 58 9 95
```

```
a :
row = 8, column = 8
7 44 2 87 80 17 23 5
60 4 35 90 68 72 79 2
83 66 41 21 76 50 59 43
33 63 90 34 7 77 34 42
47 31 21 73 18 8 44 59
82 56 0 1 86 27 37 16
93 84 30 34 60 44 92 55
70 35 43 6 39 58 9 95
a /= b, a :
row = 1, column = 1
1
```

2.3.2.2 matrix versus operands

The action of the matrix and operands is converted to the action of each element of the matrix and operands, and directly modifies the data of the original matrix. The code as follows :

```
//declaration
Matrix &operator-=(T);

Matrix &operator/=(T);

Matrix &operator*=(T);

Matrix &operator+=(T);

//defined
template<typename T>
Matrix<T> &Matrix<T>::operator/=(T a) {
    if (a == 0 || this->matrix == nullptr) {
        cerr << "DIVISION FAILURE!" << endl;
        return *this;
    }
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            this->matrix[(this->currentChannel - 1) * this->size + i * this-
>step + j] /= a;
```

```
        }
    }
    return *this;
}

template<typename T>
Matrix<T> &Matrix<T>::operator*=(T a) {
    if (this->matrix == nullptr) {
        cerr << "MUL FAILURE!" << endl;
        return *this;
    }
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            this->matrix[(this->currentChannel - 1) * this->size + i * this->step + j] *= a;
        }
    }
    return *this;
}

template<typename T>
Matrix<T> &Matrix<T>::operator+=(T a) {
    if (this->matrix == nullptr) {
        cerr << "PLUS FAILURE!" << endl;
        return *this;
    }
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            this->matrix[(this->currentChannel - 1) * this->size + i * this->step + j] += a;
        }
    }
    return *this;
}

template<typename T>
Matrix<T> &Matrix<T>::operator-=(T a) {
    if (this->matrix == nullptr) {
        cerr << "SUBTRACT FAILURE!" << endl;
        return *this;
    }
    for (int i = 0; i < this->row; i++) {
        for (int j = 0; j < this->column; j++) {
            this->matrix[(this->currentChannel - 1) * this->size + i * this->step + j] -= a;
        }
    }
    return *this;
}
```

The results are shown below :

```
Matrix<long> a = Matrix<long>(6, 6);
a.readFromFile(fileA, 1);
long tem = 6;
cout << "tem = " << tem << endl;
a /= tem;
```

```
cout << "a /= 6 : " << tem << endl;
cout << a;
/*
the test data:
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
*/
```

```
tem = 66
a *= 66 :
row = 6, column = 6
66 132 198 264 330 396
462 528 594 660 726 792
132 198 264 330 396 462
264 330 396 462 528 594
396 462 528 594 660 726
66 198 330 462 594 726

tem = 66
a += 66 :
row = 6, column = 6
67 68 69 70 71 72
73 74 75 76 77 78
68 69 70 71 72 73
70 71 72 73 74 75
72 73 74 75 76 77
67 69 71 73 75 77

tem = 66
a -= 66 :
row = 6, column = 6
-65 -64 -63 -62 -61 -60
-59 -58 -57 -56 -55 -54
-64 -63 -62 -61 -60 -59
-62 -61 -60 -59 -58 -57
-60 -59 -58 -57 -56 -55
-65 -63 -61 -59 -57 -55

tem = 6
a /= 6 : 6
row = 6, column = 6
1 7 0 14 13 2
3 0 10 0 5 15
11 12 13 0 13 11
6 3 12 8 9 7
5 10 15 5 1 12
5 7 7 5 3 12
```

2.3.3 == & != & =

Rewrite "==" and "! =", so as to judge whether the current channels of the two matrices are equal in value.

Overload "=", because the system has "=" easy to cause memory leakage, overload can effectively avoid memory leakage and maintain memory Pointers in time. ***(memory management tests for overloading "=" will be presented in "difficulty and Solution" along with constructor tests.)***

The code and test result as follows :

```
//declaration
bool operator==(const Matrix<T> &);

bool operator!=(const Matrix<T> &);

Matrix &operator=(const Matrix<T> &);

//defined
template<typename T>
bool Matrix<T>::operator==(const Matrix<T> &a) {
    if (this->row != a.row || this->column != a.column)
        return false;
    if (this->matrix == nullptr || a.matrix == nullptr)
        return false;
    for (int i = 0; i < a.row; ++i)
        for (int j = 0; j < a.column; ++j) {
            if (this->matrix[(this->currentChannel - 1) * this->size + i * this-
>step + j] !=
                a.matrix[(a.currentChannel - 1) * a.size + i * a.step + j])
                return false;
        }
    return true;
}

template<typename T>
```

```
bool Matrix<T>::operator!=(const Matrix<T> &a) {
    if (this->row != a.row || this->column != a.column)
        return true;
    if (this->matrix == nullptr || a.matrix == nullptr)
        return false;
    for (int i = 0; i < a.row; ++i)
        for (int j = 0; j < a.column; ++j) {
            if (this->matrix[(this->currentChannel - 1) * this->size + i * this->step + j] !=
                a.matrix[(a.currentChannel - 1) * a.size + i * a.step + j])
                return true;
        }
    return false;
}

//assign the corresponding variable, free the extra memory, and update the memory pointer
template<typename T>
Matrix<T> &Matrix<T>::operator=(const Matrix<T> &a) {
    if (this == &a)
        return *this;
    else {
        if (this->matrix != nullptr)
            delete[] this->matrix;
        this->matrix = a.matrix;
        this->pointTime = a.pointTime;
        this->pointTime[0]++;
        this->column = a.column;
        this->row = a.row;
        this->step = a.step;
        this->size = a.size;
        this->channel = a.channel;
        this->currentChannel = a.currentChannel;
        this->rank = a.rank;
        cout << "the address is " << this->pointTime << ", the num of point is " << this->pointTime[0] << endl;
        return *this;
    }
}
```

```
a == b ? 0    output << "a == b ? " << std::to_string(a == b) << endl;
a != b ? 1    output << "a != b ? " << std::to_string(a != b) << endl;
```

2.3.4 explicit operator T() const;

This method forces an explicit conversion of the matrix to a primitive data type and returns the mean of the current channel of the matrix. However, the data type that can be cast depends on the data type of the matrix array and must be consistent.

```
explicit operator T() const {
    return this->getAverage();
}
```

The test result as follows :

```
Matrix<long> a = Matrix<long>( r: 6,  c: 6);
a.readFromFile( in: fileA,  type: 1);
long tem = (long) a;
cout << tem;
```

```
46
```

2.4 other functions

Other matrix-related functions will be explained uniformly here.

2.4.1 code

```
// the minimum value of the current channel of the matrix can be obtained
T getMin() const;

//the maximum value of the current channel of the matrix can be obtained
T getMax() const;

//the average value of the current channel of the matrix can be obtained
T getAverage() const;

//use function to output the details of the matrix
void printMatrix() const;

//return an identity matrix of the same size as the matrix on which this method
was called
Matrix getUnit();

//return the trace of the matrix (the average of the values along the main
diagonal)
T getTrace() const;

//return the transposition of the matrix
Matrix getTranspose() const;

//the average value a row of the matrix
T getRowAve(int i) const;

//the average value a column of the matrix
T getColumnAve(int i) const;

//remove a row of the matrix
Matrix removeRow(int) const;

//remove a column of the matrix
Matrix removeColumn(int) const;

//change the row and column of the matrix
void changeSize(int i, int j);

//output the matrix to the file
void outputToFile(std::ofstream &);

//read the matrix from the file
void readFromFile(const string &, int);
```

```
//LU factorization of matrices gives the upper and lower triangular matrices
Matrix LU(Matrix<T> &) const;

//change two rows of the matrix
void swapRow(int, int);

//multiply a row by a number
void scaleRow(int, T);

//get the determinant of the matrix
T getDet() const;

//get the inverse of the matrix
Matrix inverse();

//get the rank and the ladder matrix
Matrix getRankLadder();

//get the adjoint matrix
Matrix getAdjoin() const {
    return this->inverse() * this->getDet();
}
```

2.4.2 node

You can change the type of data to be read by modifying the parameters. (**warning** : The same type as declared in the matrix.)

```
template<typename T>
void Matrix<T>::readFromFile(const string &in, int type) {
    try {
        ifstream infile;
        infile.open(in);
        std::string t;
        int count = 0;
        while (count < this->channel) {
            for (int i = 0; i < this->row; ++i)
                for (int j = 0; j < this->column; ++j) {
                    infile >> t;
                    //1-int 2-long 3-unsigned-long 4-l-l 5-un-l-l 6-float 7-
double 8-long-double
                    switch (type) {
                        case 1:
                            this->matrix[i * this->column + j] = stoi(t);
                            break;
                        case 2:
                            this->matrix[i * this->column + j] = stol(t);
                            break;
                        case 3:
                            this->matrix[i * this->column + j] = stoul(t);
                            break;
                        case 4:
                            this->matrix[i * this->column + j] = stoll(t);
                            break;
                        case 5:
                            this->matrix[i * this->column + j] = stoull(t);
                            break;
```

```
                        case 6:
                            this->matrix[i * this->column + j] = stof(t);
                            break;
                        case 7:
                            this->matrix[i * this->column + j] = stod(t);
                            break;
                        case 8:
                            this->matrix[i * this->column + j] = stold(t);
                            break;
                        default:
                            this->matrix[i * this->column + j] = stod(t);
                            break;
                    }
                }
            count++;
        }
        infile.close();
    } catch (int exception) {
        cerr << "ERROR" << endl;
    } catch (double) {
        cerr << "ERROR" << endl;
    }
}
```

2.4.3 example

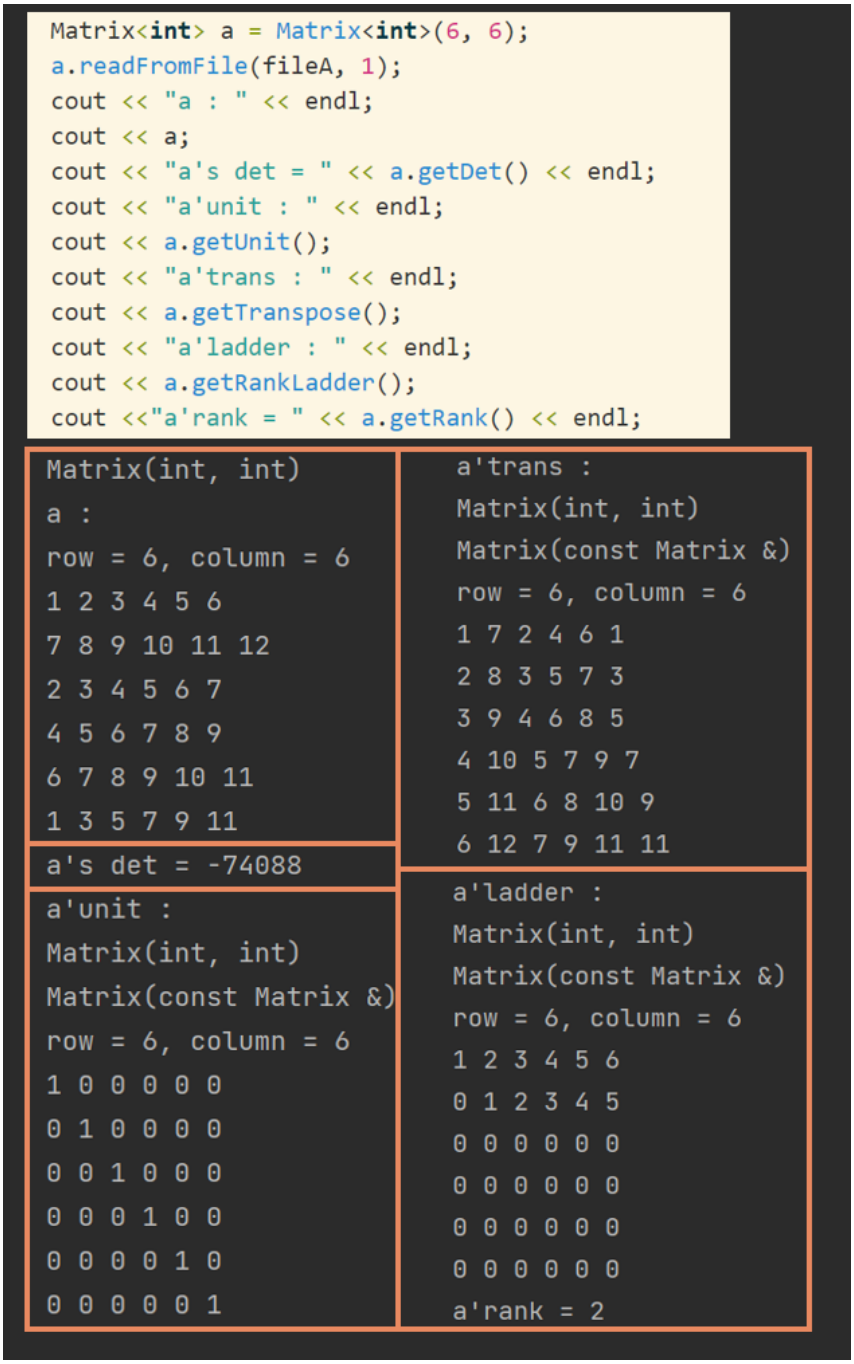
I will use some examples to show the results of these methods. (**warning** : the extra constructor calls are temporary variables inside the method.)

① The following method declarations are used in this test.

```
Matrix getUnit();
Matrix getTranspose() const;
T getDet() const;
Matrix getRankLadder();
void readFromFile(const string &, int);
```

The test code and its results are shown below.

Each red box boxes the execution of a method, which also contains the output of the constructor call. Execution results are presented from left to right.

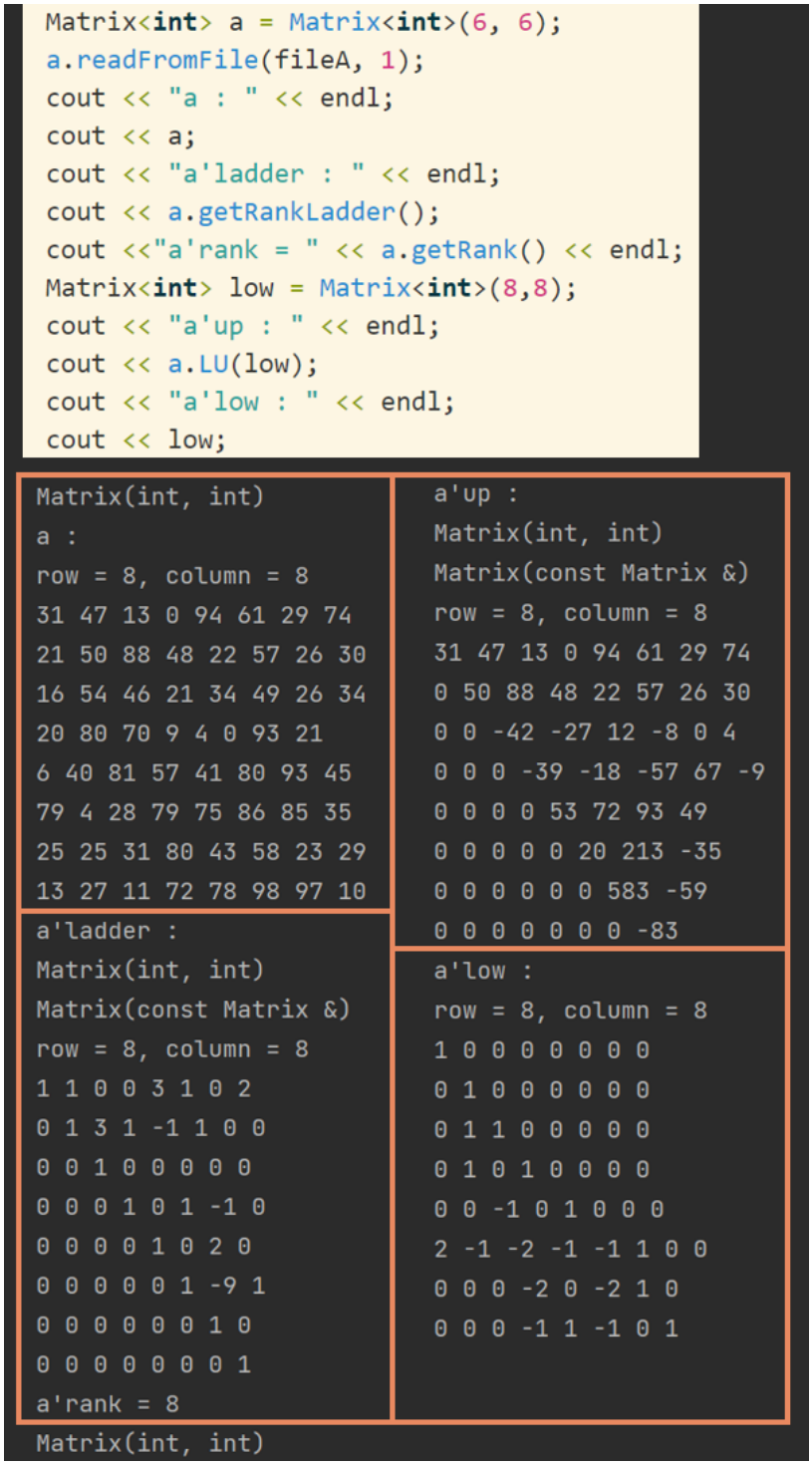


②The following method declarations are used in this test.

```
void readFromFile(const string &, int);
Matrix LU(Matrix<T> &) const;
Matrix getRankLadder();
```

The test code and its results are shown below.

Each red box boxes the execution of a method, which also contains the output of the constructor call. Execution results are presented from left to right.



③The following method declarations are used in this test.

```
void readFromFile(const string &, int);
Matrix inverse();
Matrix getRankLadder();
```

The test code and its results are shown below.

Each red box boxes the execution of a method, which also contains the output of the constructor call.

```
Matrix(int, int)
a :
row = 8, column = 8
7.800000 44.400000 2.300000 87.700000 80.100000 17.900000 23.100000 5.300000
60.900000 4.300000 35.900000 90.600000 68.200000 72.200000 79.400000 2.700000
83.500000 66.300000 41.700000 21.500000 76.700000 50.100000 59.700000 43.100000
33.400000 63.000000 90.200000 34.600000 7.100000 77.800000 34.500000 42.400000
47.000000 31.000000 21.200000 73.700000 18.000000 8.600000 44.900000 59.400000
82.100000 56.900000 0.000000 1.900000 86.400000 27.900000 37.600000 16.800000
93.700000 84.800000 30.300000 34.800000 60.700000 44.300000 92.600000 55.200000
70.100000 35.600000 43.100000 6.500000 39.700000 58.600000 9.800000 95.000000
```

```
Matrix(int, int)
Matrix(const Matrix &)
a' inverse :
Matrix(int, int)
Matrix(const Matrix &)
```

```
Matrix<double> a = Matrix<double>(8, 8);
a.readFromFile(fileA, 0);
cout << "a : " << endl;
cout << a;
a.getRankLadder();
cout << "a' inverse : " << endl;
cout << a.inverse();
```

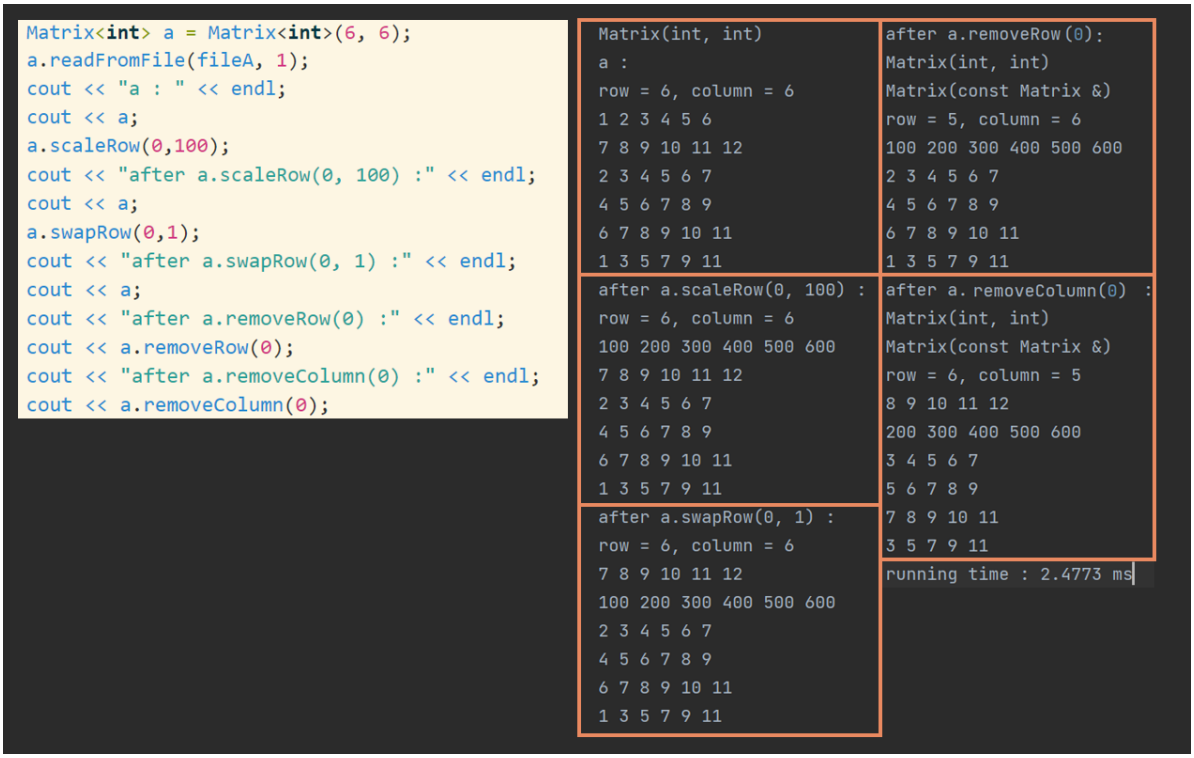
```
row = 8, column = 8
-0.015456 0.003003 -0.018074 0.011122 0.022070 0.037071 -0.016233 -0.006910
0.005137 -0.007444 -0.023668 0.011870 -0.000633 0.015396 0.010306 -0.002950
-0.006646 -0.004606 0.054357 0.000770 0.012110 -0.015374 -0.027967 -0.013106
0.001403 0.002665 -0.018345 0.007460 0.012943 0.014673 -0.005543 -0.002627
0.007857 -0.001624 0.036922 -0.012902 -0.007997 -0.019727 -0.007853 0.001667
0.004930 0.009528 -0.045851 0.007704 -0.019197 0.010564 0.020732 0.014906
0.001548 0.002568 0.021158 -0.013567 -0.012610 -0.030964 0.018423 -0.001048
0.005915 -0.002983 0.009471 -0.011476 -0.005943 -0.022231 0.009777 0.013073
```

④The following method declarations are used in this test.

```
Matrix removeRow(int) const;
Matrix removeColumn(int) const;
void readFromFile(const string &, int);
void swapRow(int, int);
void scaleRow(int, T);
Matrix inverse();
```

The test code and its results are shown below.

Each red box boxes the execution of a method, which also contains the output of the constructor call. Execution results are presented from left to right. The last one shows the running time, is 2.4773ms.



⑤The following method declarations are used in this test.

```
T getRowAve(int i) const;
T getColumnAve(int i) const;
void changeSize(int i, int j);
void readFromFile(const string &, int);
```

The test code and its results are shown below.

Each red box boxes the execution of a method, which also contains the output of the constructor call.

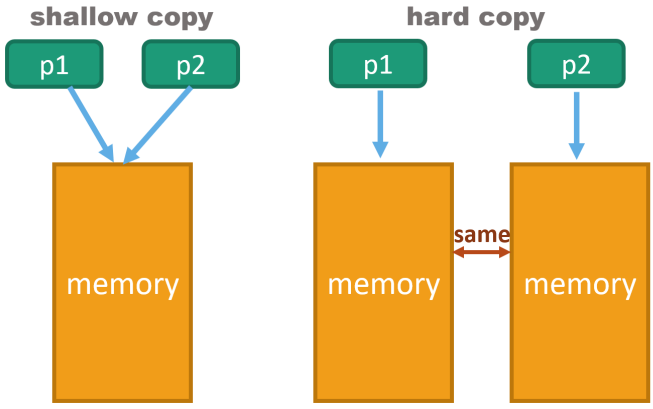
```
Matrix<long> a = Matrix<long>( r: 6,  c: 6);
a.readFromFile( in: fileA,  type: 1);
cout << "a : " << endl;
cout << a;
a.changeSize( i: 4, j: 9);
cout << "a.changeSize(4, 9) : " << endl;
cout << a;
cout << "a.getRowAve(0) = " << a.getRowAve( i: 0) << endl;
cout << "a.getColumnAve(0) = " << a.getColumnAve( i: 0) << endl;
Matrix(int, int)
a :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
a.changeSize(4, 9) :
row = 4, column = 9
1 2 3 4 5 6 7 8 9
10 11 12 2 3 4 5 6 7
4 5 6 7 8 9 6 7 8
9 10 11 1 3 5 7 9 11
a.getRowAve(0) = 5
a.getColumnAve(0) = 6
```

3 Difficulty & Solution

3.1 memory management

3.1.1 shallow copy & deep copy

Shallow copy copies only Pointers to an object, not the object itself, and the old and new objects still share the same memory. But deep copy will create another identical object, the new object and the original object do not share memory, modify the new object will not change to the original object. The graph shows as follows :



3.1.2 why shallow copy

The base data type is passed by value, and the reference data type is copied by reference, which is a shallow copy. A shallow copy is a bitwise copy of an object that creates a new object with an exact copy of the original object's property values. If the property is of a primitive type, the value of the primitive type is copied. If the property is a memory address (reference type), the memory address is copied, so if one object changes the address, the other object will be affected. **That is, the default copy constructor only copies objects shallowly (member by member), that is, only the object space is copied, not the resource.**

Most of the time there are only read-only operations on matrices. OpenCV, for example, to transform it into image capture in the form of matrix, and most of the time only read-only operations, and didn't have a lot of write operation, in this case, the large-scale data if every copy is deep copy, will waste a lot of memory space, serious can cause stack overflow, so use shallow copy.

3.1.3 memory analysis

3.1.3.1 copy constructor & assignment operation

The array requests memory from the system in the form of **new**, and maintains an **int * pointer** and allocates 4 bits of space for it to store the number of times the memory address is pointed to.

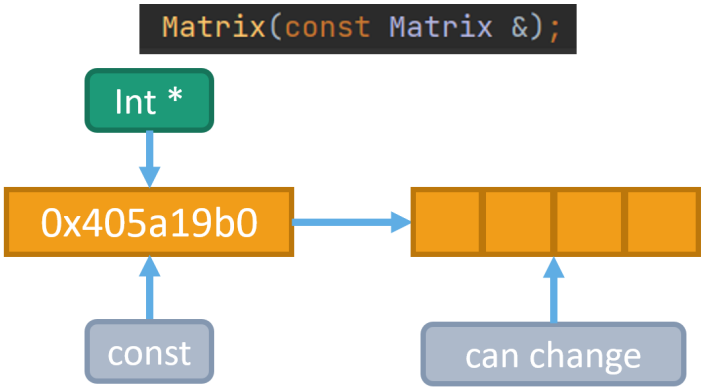
① why not use int to record data directly?

Since the copy constructor is overwritten, the parameters passed in are required to be "const", that is, no data from the matrix passed as parameters can be directly modified inside the overwritten method body, which severely affects memory management. Therefore, this method is not feasible.

②why the int * pointer can work?

When I overwrite the copy constructor, I require that the argument passed in be "const". If I use an int * pointer, I cannot modify the address stored by the pointer itself, but I can operate on the memory to which it points, so the problem of not maintaining a memory counter is solved perfectly.

The graph shows below :



③how to implement

The total output and code are shown below :

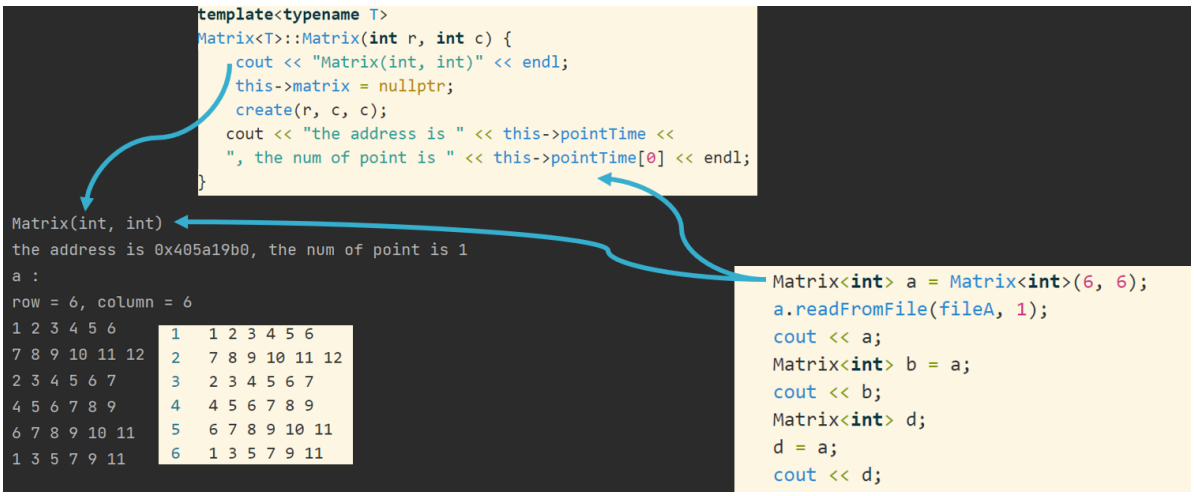
```
Matrix(int, int)
the address is 0x405a19b0, the num of point is 1
a :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
Matrix(const Matrix &)
the address is 0x405a19b0, the num of point is 2
b :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
Matrix()
the address is 0x405a19b0, the num of point is 3
d :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
pointer = 0x1043fb60, matrix pointer = 0x405a19d0, pointTime = 3
pointer = 0x1043fb90, matrix pointer = 0x405a19d0, pointTime = 2
pointer = 0x1043fbc0, matrix pointer = 0x405a19d0, pointTime = 1
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

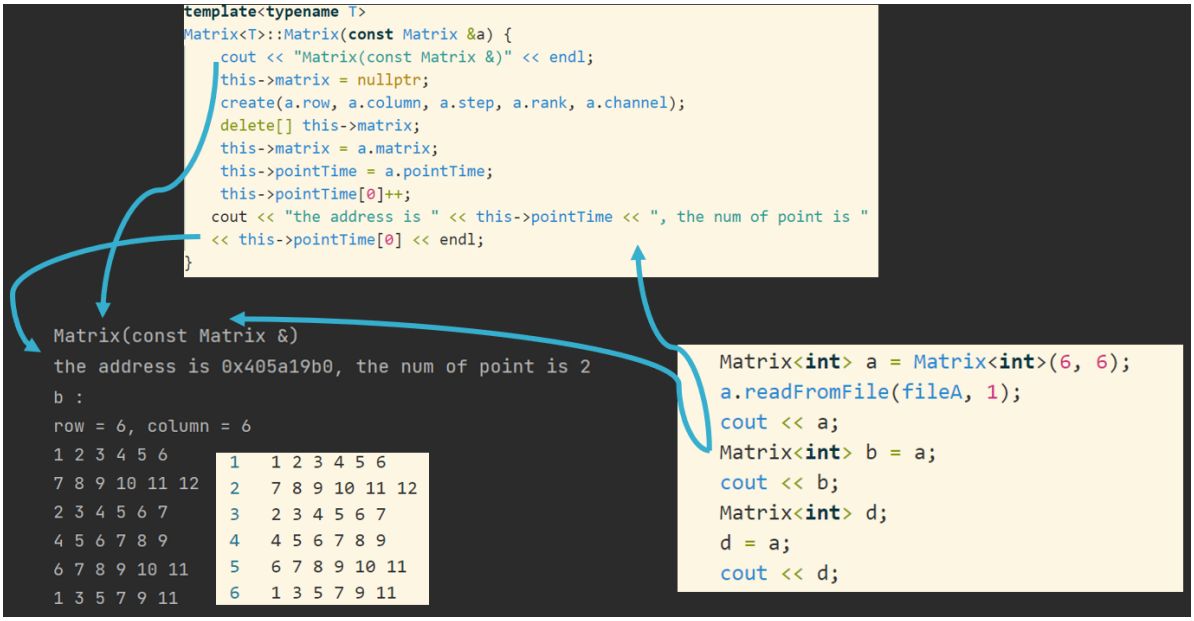
```
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> b = a;
cout << b;
Matrix<int> d;
d = a;
cout << d;
```

The details are as follows :

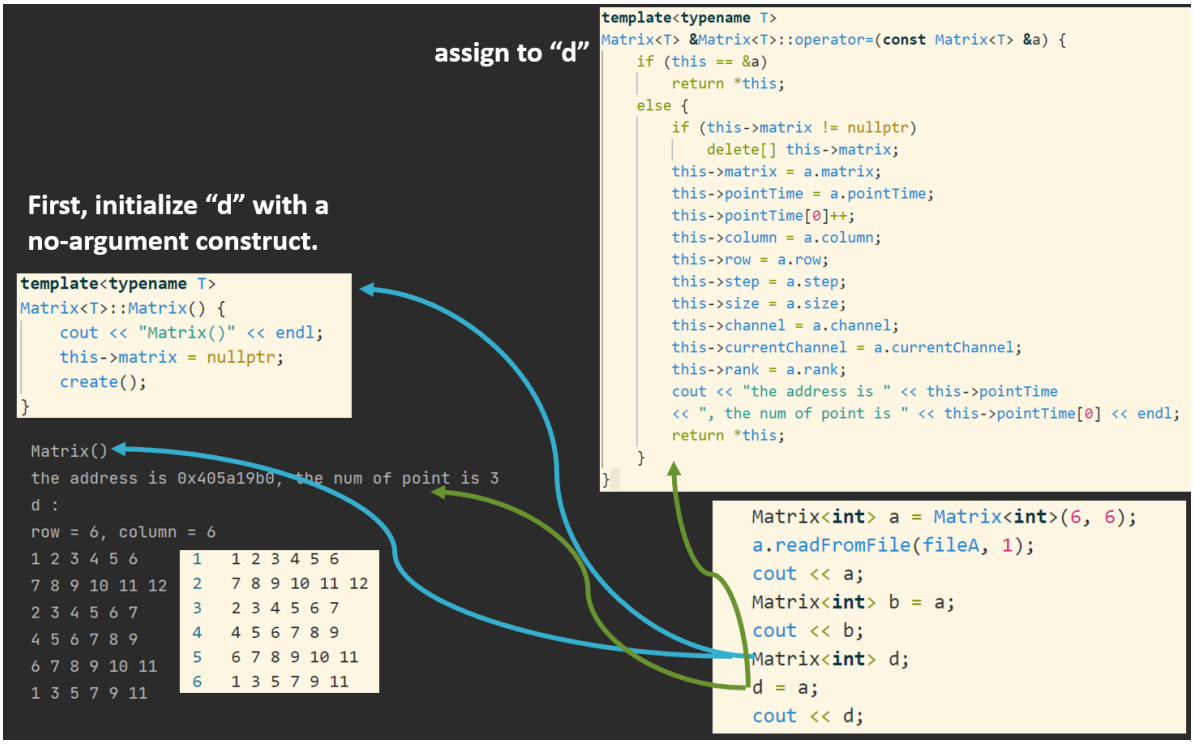
initialize "a" matrix, as follows :



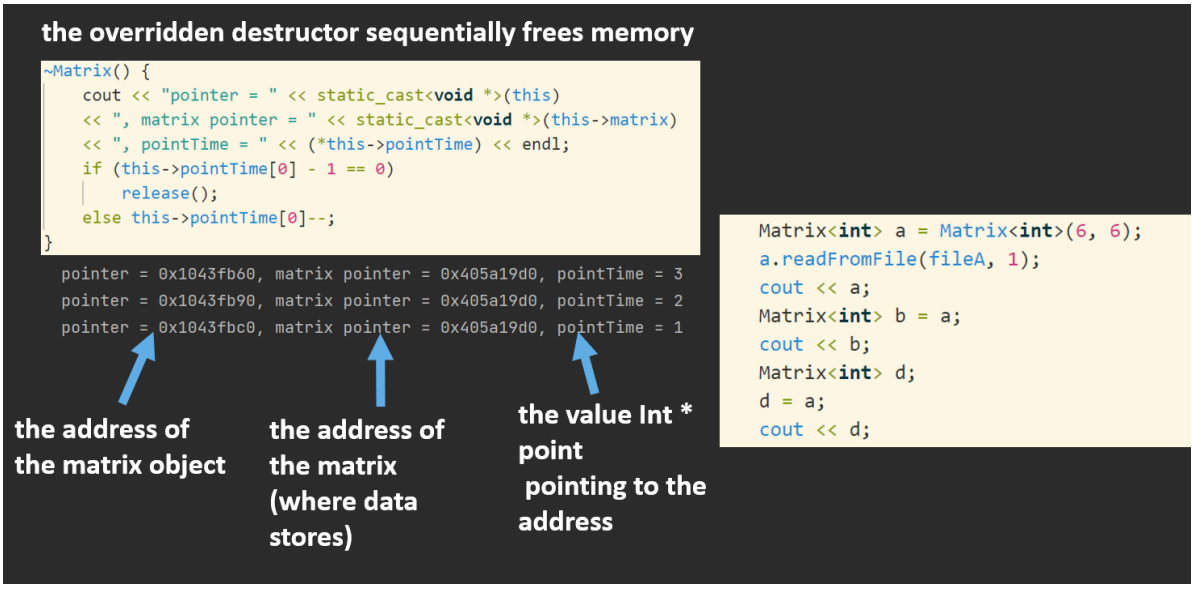
initialize "b" matrix, as follows :



initialize and assign "d" matrix, as follows :



destructor called :



3.1.3.2 ROI

ROI is designed to be able to capture the desired sub-matrices from a large matrix and share them with large matrices through shallow copies.

The total output and code are shown below :

```
Matrix(int, int)
the address is 0x408719b0, the num of point is 1
a :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871aa0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x408719d0, pointTime = 3
c (the submatrix of a) :
row = 3, column = 3
1 2 3
7 8 9
2 3 4
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871ae0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x40871a24, pointTime = 3
d (the submatrix of a) :
row = 3, column = 3
7 8 9
9 10 11
7 9 11
pointer = 0x1043fb60, matrix pointer = 0x40871a24, pointTime = 2
pointer = 0x1043fb90, matrix pointer = 0x408719d0, pointTime = 2
pointer = 0x1043fbc0, matrix pointer = 0x408719d0, pointTime = 1
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

```
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << d;
```

The details are as follows :

initialize "a" matrix, as follows :

```cpp
template<typename T>
Matrix<T>::Matrix(int r, int c) {
    cout << "Matrix(int, int)" << endl;
    this->matrix = nullptr;
    create(r, c, c);
    cout << "the address is " << this->pointTime <<
    ", the num of point is " << this->pointTime[0] << endl;
}
```

```
Matrix(int, int)
the address is 0x408719b0, the num of point is 1
a :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

```cpp
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> b = a;
cout << b;
Matrix<int> d;
d = a;
cout << d;
```

The constructor is called inside the method body.

```cpp
template<typename T>
Matrix<T> Matrix<T>::getSubMat(int i, int j, int r, int c) {
    if (this->matrix == nullptr) {
        cerr << "ERROR" << endl;
        return *this;
    }
    if (i < 0 || j < 0 || i > this->row - 1 || j > this->column - 1) {
        cerr << "POINT OUT OF MATRIX" << endl;
        return *this;
    }
    if (r < 0 || c < 0 || r > this->row || c > this->column ||
    r + i > this->row || j + c > this->column) {
        cerr << "SIZE OUT OF LIMITATION" << endl;
        return *this;
    }
    T *t = this->matrix + this->size * (this->currentChannel - 1) + i * this->step + j;
    Matrix<T> res = Matrix<T>(r, c, t);
    res.pointTime[0] = this->pointTime[0] + 1;
    res.step = this->column;
    return res;
}
```

```cpp
template<typename T>
Matrix<T>::Matrix(int r, int c, T *a) {
    cout << "Matrix(int, int, T *)" << endl;
    this->matrix = nullptr;
    create(r, c, c);
    delete[] this->matrix;
    this->matrix = a;
}
```

```
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871aa0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x408719d0, pointTime = 3
c (the submatrix of a) :
row = 3, column = 3
1 2 3
7 8 9
2 3 4
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

```cpp
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << d;
```

The overwritten constructor is called.

```cpp
template<typename T>
Matrix<T>::Matrix(const Matrix &a) {
    cout << "Matrix(const Matrix &)" << endl;
    this->matrix = nullptr;
    create(a.row, a.column, a.step, a.rank, a.channel);
    delete[] this->matrix;
    this->matrix = a.matrix;
    this->pointTime = a.pointTime;
    this->pointTime[0]++;
    cout << "the address is " << this->pointTime << ", the num of point is "
    << this->pointTime[0] << endl;
}
```

```
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871aa0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x408719d0, pointTime = 3
c (the submatrix of a) :
row = 3, column = 3
1 2 3
7 8 9
2 3 4
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

```cpp
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << d;
```

a's matrix address : 0x408719d0
c's matrix address : 0x408719d0

They are the same, share the same memory

start at (0,0), take a 3 by 3 matrix

```
pointer matrix = 0x408719d0
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871aa0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x408719d0, pointTime = 3
c (the submatrix of a) :
row = 3, column = 3
1 2 3
7 8 9
2 3 4
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

```
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << d;
```

```
template<typename T>
Matrix<T> Matrix<T>::getSubMat(int i, int j, int r, int c) {
    if (this->matrix == nullptr) {
        cerr << "ERROR" << endl;
        return *this;
    }
    if (i < 0 || j < 0 || i > this->row - 1 || j > this->column - 1) {
        cerr << "POINT OUT OF MATRIX" << endl;
        return *this;
    }
    if (r < 0 || c < 0 || r > this->row || c > this->column ||
    r + i > this->row || j + c > this->column) {
        cerr << "SIZE OUT OF LIMITATION" << endl;
        return *this;
    }
    T *t = this->matrix + this->size * (this->currentChannel - 1) + i * this->step + j;
    Matrix<T> res = Matrix<T>(r, c, t);
    res.pointTime[0] = this->pointTime[0] + 1;
    res.step = this->column;
    return res;
}
```

it's going to be 2, because "res" will be deleted

```
pointer matrix = 0x408719d0
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871aa0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x408719d0, pointTime = 3
c (the submatrix of a) :
row = 3, column = 3
1 2 3
7 8 9
2 3 4
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

```
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << d;
```

a's matrix address : 0x408719d0
d's matrix address : 0x40871a24

0x408719d0 - 0x40871a24
= 84 bits = 4 bits * 21

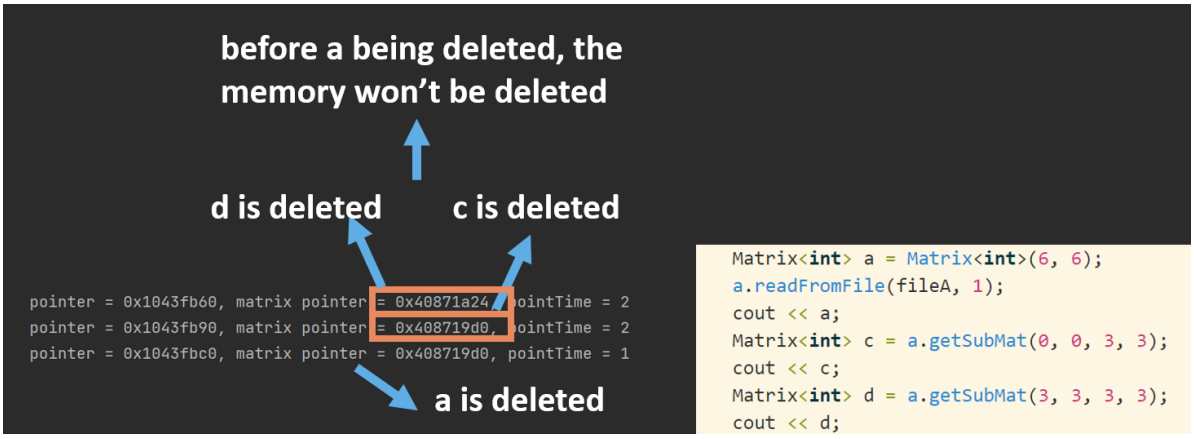
share the same memory

start at (3,3), take a 3 by 3 matrix

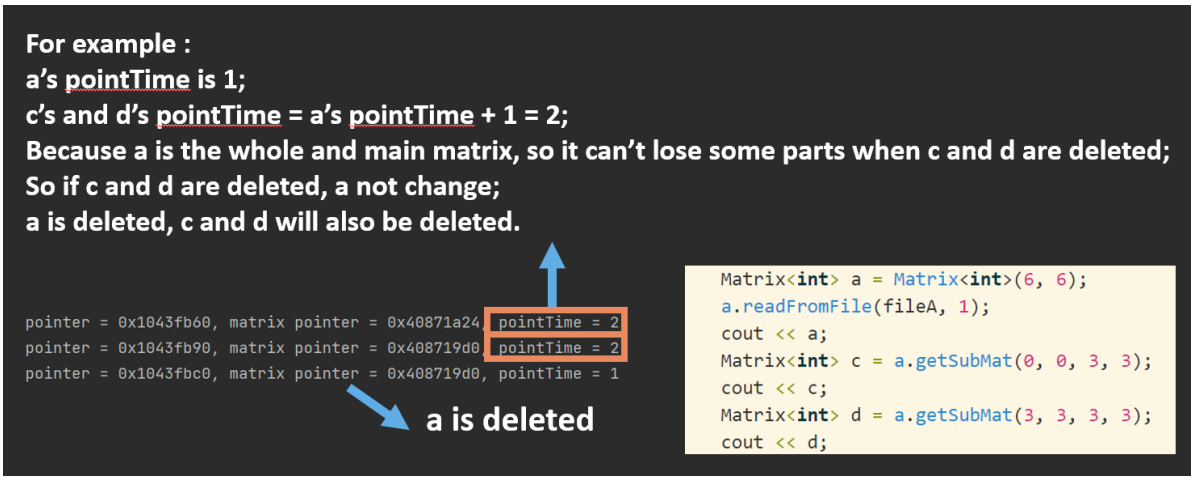
```
Matrix(int, int, T* &)
Matrix(const Matrix &)
the address is 0x40871ae0, the num of point is 3
pointer = 0x1043fad0, matrix pointer = 0x40871a24, pointTime = 3
d (the submatrix of a) :
row = 3, column = 3
7 8 9
9 10 11
7 9 11
```

```
1   1 2 3 4 5 6
2   7 8 9 10 11 12
3   2 3 4 5 6 7
4   4 5 6 7 8 9
5   6 7 8 9 10 11
6   1 3 5 7 9 11
```

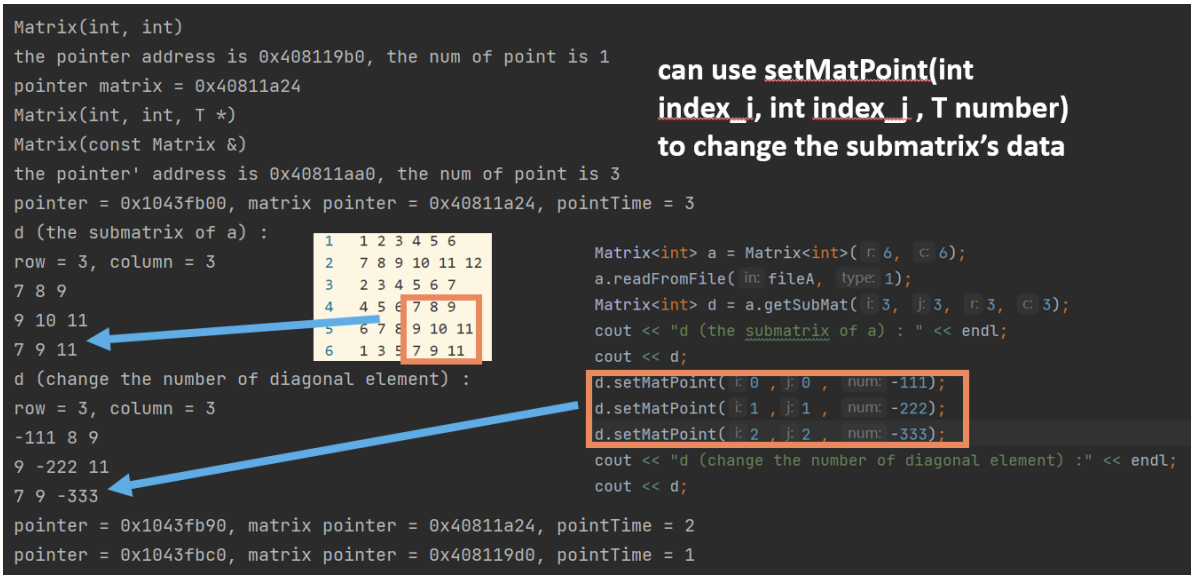
```
Matrix<int> a = Matrix<int>(6, 6);
a.readFromFile(fileA, 1);
cout << a;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << d;
```



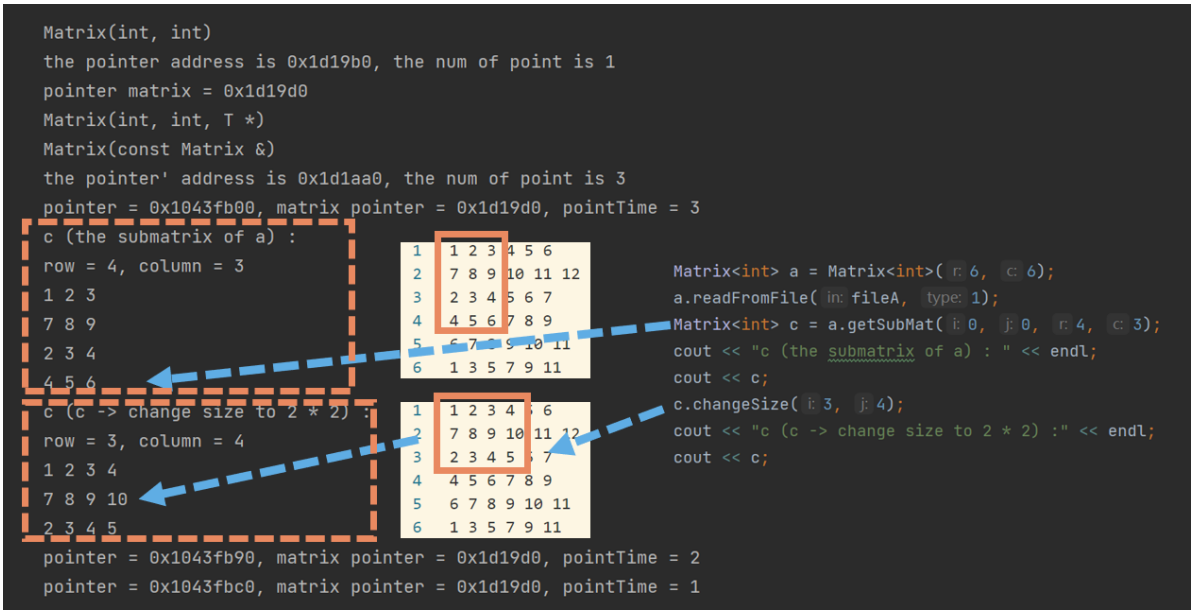
for the pointTime :



You can modify the ROI value :



You can modify the matrix size, but **be careful of memory leaks**.



3.2 generic class

A template class to allow the user to define a pattern, makes some of the data members of the class, the default member function parameters, some member function return values, to be able to take any type (including the system of predefined and user-defined), it not only represents a specific, practical classes, but represents a kind of class.

①why use template class

Because the matrix class needs to support a variety of basic data types, the template class is adopted for users to use, and users can choose which basic data types or even read the matrix data according to their needs.

②why not flag (as the same as OpenCV)

First, template classes can cover a wide range of data types including basic data types and other data types (if it is user-defined, you need to overload the operators of the user-defined data types before using the template class), but **flag** can only contain a limited number of data types.

Maybe using **flag** can make the program more efficient, however, I still want to try to implement matrix classes in the same way as template classes.

3.3 ARM & x86

3.3.1 complier

3.3.1.1 difference

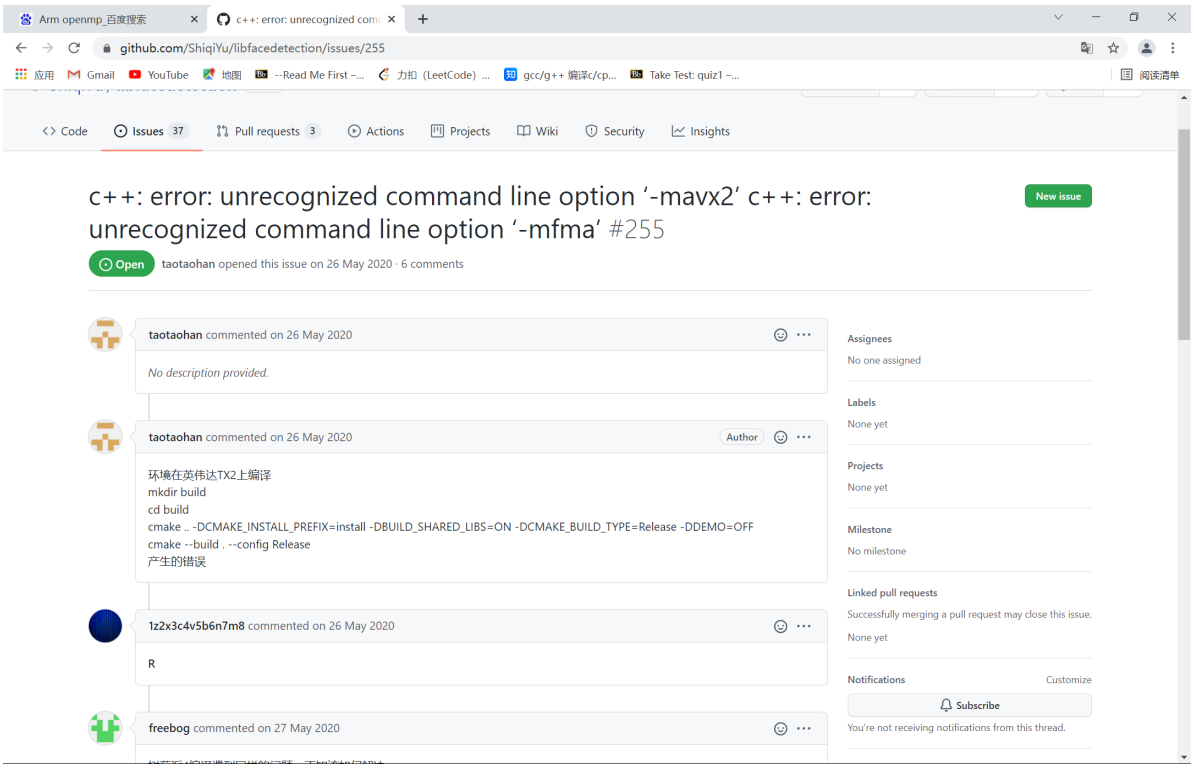
①x86 compilers can support compiler optimizations, such as turning on compiler optimizations in cmakelist.txt.

```
add_compile_options(-mavx2)
```

However, the ARM compiler does not support compiler optimization, and it will display an error message if compiler optimization is turned on, such as

```
[root@ecs001-0006 pro04]# make
Scanning dependencies of target mat
[ 50%] Building CXX object CMakeFiles/mat.dir/main.cpp.o
c++: error: unrecognized command line option ‘-mavx2’
make[2]: *** [CMakeFiles/mat.dir/build.make:63: CMakeFiles/mat.dir/main.cpp.o] Error 1
make[1]: *** [CMakeFiles/Makefile2:73: CMakeFiles/mat.dir/all] Error 2
make: *** [Makefile:84: all] Error 2
```

supporting evidence as follows :



ShiqiYu commented on 27 May 2020 — Owner

ARM不支持AVX，所以不能用这些选项。

...

> On Wed, May 27, 2020 at 3:23 PM freebog ***@***.***> wrote:
> 树莓派4编译遇到同样的问题，不知该如何解决
>
> —
> You are receiving this because you are subscribed to this thread.
> Reply to this email directly, view it on GitHub
> <#255 (comment)>,
> or unsubscribe
> <https://github.com/notifications/unsubscribe-auth/ABWR4HLM6J75Y4P7W3P2ETTRTS5W7ANCNFSM4NJ4G25Q>
> .

> --
>
> Prof. Shiqi YU (于仕琪)
>
> Department of Computer Science and Engineering,
>
> Southern University of Science and Technology,
>
> Shenzhen, China.

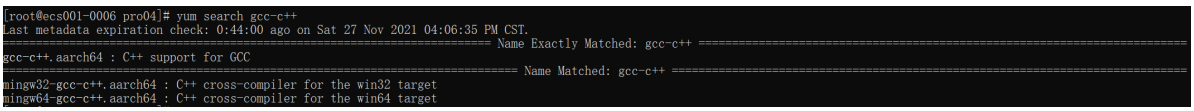
②The compiler is constructed differently, for x86:

it can support version 3.16, even 3.20;

```
cmake_minimum_required(VERSION 3.16)
```

```
cmake_minimum_required(VERSION 3.20)
```

for ARM:



after enlarged :

```
[root@ecs001-0006 pro04]# yum search gcc-c++
Last metadata expiration check: 0:44:00 ago on Sat 27 Nov 2021 04:06:35 PM CST.
================================================================================ Name Exactly Matched: gcc-c++
gcc-c++.aarch64 : C++ support for GCC
================================================================================ Name Matched: gcc-c++ ====
mingw32-gcc-c++.aarch64 : C++ cross-compiler for the win32 target
mingw64-gcc-c++.aarch64 : C++ cross-compiler for the win64 target
```

The c++ compiler on ARM is **gcc-c++.aarch.64**. It can support version 3.12.1;

```
[root@ecs001-0006 pro04]# cat CMakeLists.txt
cmake_minimum_required(VERSION 3.12.1)
```

3.3.1.2 same

①It can also be compiled using cmake or directly by command.

for x86:

```
linjiefang@LAPTOP-99UCO9DE:~/project04$ cmake .
OPENMP FOUND
-- Configuring done
-- Generating done
-- Build files have been written to: /home/linjiefang/project04
linjiefang@LAPTOP-99UCO9DE:~/project04$ make
[ 50%] Building CXX object CMakeFiles/mat.dir/main.cpp.o
[100%] Linking CXX executable mat
[100%] Built target mat
```

for ARM:

```
[root@ecs001-0006 pro04]# cmake .
OPENMP FOUND
-- Configuring done
-- Generating done
-- Build files have been written to: /home/linjiefang/cppPro/pro04
[root@ecs001-0006 pro04]# make
[100%] Built target mat
```

②They can also use OpenMP to optimize multiplications.

for x86:

```
FIND_PACKAGE(OpenMP REQUIRED)
if (OPENMP_FOUND)
    message("OPENMP FOUND")
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
endif ()

add_compile_options(-fopenmp)
```

for ARM:

```
FIND_PACKAGE(OpenMP REQUIRED)
if (OPENMP_FOUND)
    message("OPENMP FOUND")
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
endif ()

# add_compile_options(-mavx2)
add_compile_options(-fopenmp)
```

3.3.2 program running

3.3.2.1 difference

①about OpenMP

run the same code as follows :

```
Matrix<int> a = Matrix<int>(2048, 2048);
a.readFromFile(fileA, 1);
Matrix<int> b = Matrix<int>(2048, 2048);
b.readFromFile(fileB, 1);
Matrix<int> c = a * b;
c.outputToFile(output);
```

for x86:

Under x86 system, using OpenMP has a relatively significant improvement in efficiency. It's roughly doubled.

not compiler optimization

```
root@LAPTOP-99UCO9DE:/home/linjiefang/project04# ./mat 20481.txt 20482.txt res.txt
Matrix(int, int)
Matrix(int, int)
Matrix(int, int)
Matrix(const Matrix &)
pointer = 0x7fffdbab4550, matrix pointer = 0x7f91500e0010, pointTime = 2
running time : 46503.4 ms
pointer = 0x7fffdbab4660, matrix pointer = 0x7f91500e0010, pointTime = 1
pointer = 0x7fffdbab4630, matrix pointer = 0x7f91510f0010, pointTime = 1
pointer = 0x7fffdbab4600, matrix pointer = 0x7f9152100010, pointTime = 1
```

not OpenMP

```
root@LAPTOP-99UCO9DE:/home/linjiefang/project04# ./mat 20481.txt 20482.txt res.txt
Matrix(int, int)
Matrix(int, int)
Matrix(int, int)
Matrix(const Matrix &)
pointer = 0x7fffe349e2f0, matrix pointer = 0x7fbf55440010, pointTime = 2
running time : 19837.9 ms
pointer = 0x7fffe349e400, matrix pointer = 0x7fbf55440010, pointTime = 1
pointer = 0x7fffe349e3d0, matrix pointer = 0x7fbf56450010, pointTime = 1
pointer = 0x7fffe349e3a0, matrix pointer = 0x7fbf57460010, pointTime = 1
```

OpenMP

for ARM:

There is no obvious efficiency improvement. The difference is about 20 seconds.

not compiler optimization

```
[root@ecs001-0006 pro04]# ./mat mat-A-2048.txt mat-B-2048.txt res.txt
Matrix(int, int)
Matrix(int, int)
Matrix(int, int)
Matrix(const Matrix &)
pointer = 0xfffff093bae8, matrix pointer = 0xfffd3b200010, pointTime = 2
running time : 80254.9 ms
pointer = 0xfffff093bb80, matrix pointer = 0xfffd3b200010, pointTime = 1
pointer = 0xfffff093bbb0, matrix pointer = 0xfffd3c210010, pointTime = 1
pointer = 0xfffff093bbe0, matrix pointer = 0xfffd3d220010, pointTime = 1
```

OpenMP

```
[root@ecs001-0006 pro04]# ./mat mat-A-2048.txt mat-B-2048.txt res.txt
Matrix(int, int)
Matrix(int, int)
Matrix(int, int)
Matrix(const Matrix &)
pointer = 0xffffdd66b640, matrix pointer = 0xfffbebdb0010, pointTime = 2
running time : 99130.9 ms
pointer = 0xffffdd66b6c0, matrix pointer = 0xfffbebdb0010, pointTime = 1
pointer = 0xffffdd66b6f0, matrix pointer = 0xfffbecdc0010, pointTime = 1
pointer = 0xffffdd66b720, matrix pointer = 0xfffbeddd0010, pointTime = 1
```

not OpenMP

In summary, the lack of compiler optimization leads to low program efficiency, and ARM itself is slightly less efficient than x86.

the test code and result as follows :

```
Matrix<int> a = Matrix<int>(256, 256);
a.readFromFile(fileA, 1);
Matrix<int> b = Matrix<int>(256, 256);
b.readFromFile(fileB, 1);
Matrix<int> c = a * b;
c.outputToFile(output);
```

```
linjiefang@LAPTOP-99UCO9DE:~/project04$ ./mat 2561.txt 2562.txt res.txt
Matrix(int, int)
Matrix(int, int)
Matrix(int, int)
Matrix(const Matrix &)
pointer = 0x7fffc7ccf6e0, matrix pointer = 0x7f06f79b0010, pointTime = 2
running time : 97.1356 ms
pointer = 0x7fffc7ccf7f0, matrix pointer = 0x7f06f79b0010, pointTime = 1
pointer = 0x7fffc7ccf7c0, matrix pointer = 0x7f06f7a00010, pointTime = 1
pointer = 0x7fffc7ccf790, matrix pointer = 0x7f06f7a50010, pointTime = 1
```

```
[root@ecs001-0006 pro04]# ./mat 1.txt 2.txt res.txt
Matrix(int, int)
Matrix(int, int)
Matrix(int, int)
Matrix(const Matrix &)
pointer = 0xffffff3b83b68, matrix pointer = 0xfffc9bdb0010, pointTime = 2
running time : 178.4 ms
pointer = 0xffffff3b83c00, matrix pointer = 0xfffc9bdb0010, pointTime = 1
pointer = 0xffffff3b83c30, matrix pointer = 0xfffc9be00010, pointTime = 1
pointer = 0xffffff3b83c60, matrix pointer = 0xfffc9be50010, pointTime = 1
```

For both systems, deviations from the correct results occurred when OpenMP multi-parallel execution programs were opened.

However, the x86 system has many cores, so in the test of this project, the opening and closing of OpenMP does not affect the correctness of the answer. As for ARM system, it will get different results every time when OpenMP is on, and the correct results can be obtained only when OpenMP is closed.

for x86:

THE SAME

not OpenMP

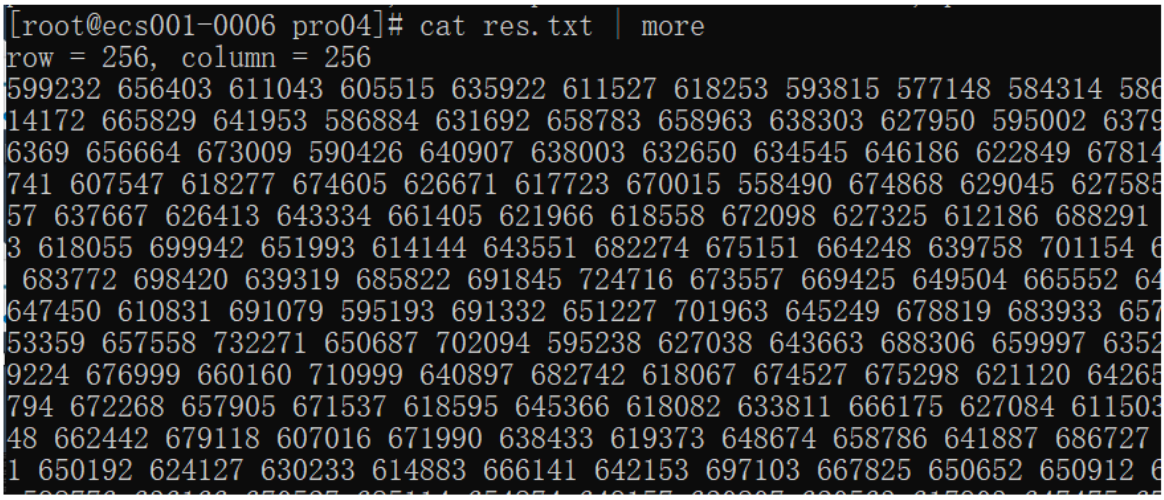
```
row = 256, column = 256
599232 656403 611043 608677 637911 615928 617450 595251
611097 682126 619074 602773 651077 628541 609874 621318
610422 681809 631681 640808 653347 632937 618531 649750
660203 672798 609820 651823 694360 654829 618189 648368
569846 615236 556038 587180 614416 603664 576846 611941
565114 638468 570648 577210 586790 595040 570832 587647
590997 665318 606192 619658 640336 598173 622957 615196
597076 623016 587281 593062 627915 604969 560822 593945
```

OpenMP

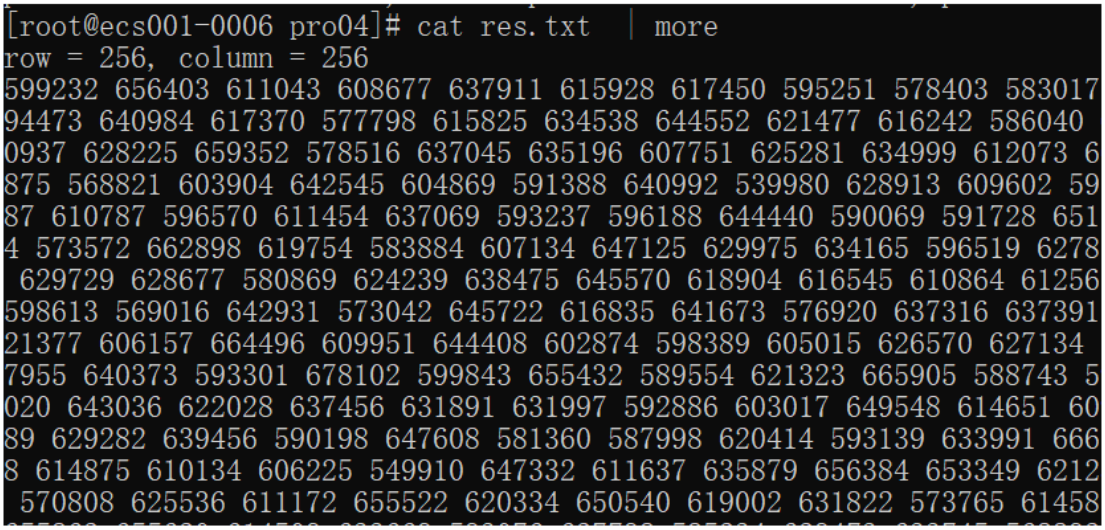
```
row = 256, column = 256
599232 656403 611043 608677 637911 615928 617450
611097 682126 619074 602773 651077 628541 609874
610422 681809 631681 640808 653347 632937 618531
660203 672798 609820 651823 694360 654829 618189
569846 615236 556038 587180 614416 603664 576846
565114 638468 570648 577210 586790 595040 570832
590997 665318 606192 619658 640336 598173 622957
597076 623016 587281 593062 627915 604969 560822
```

for ARM:

DIFFERENCE
not OpenMP



OpenMP



②about address

The address assigned to the dynamic application is different.

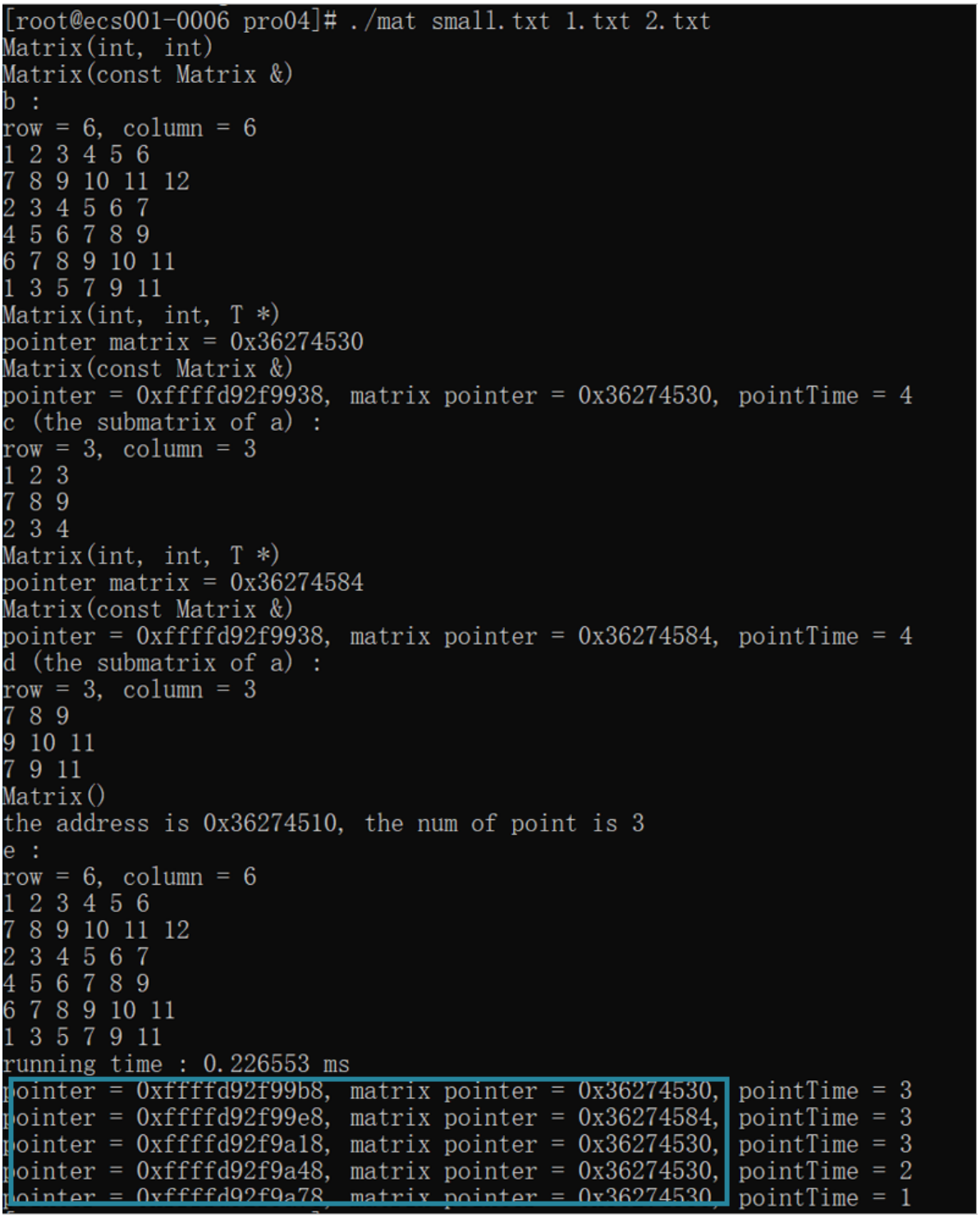
the test code below here :

```
 Matrix<int> a = Matrix<int>(6, 6);
 a.readFromFile(fileA, 1);
 Matrix<int> b = a;
cout << "b : " << endl;
cout << b;
Matrix<int> c = a.getSubMat(0, 0, 3, 3);
cout << "c (the submatrix of a) : " << endl;
cout << c;
Matrix<int> d = a.getSubMat(3, 3, 3, 3);
cout << "d (the submatrix of a) : " << endl;
cout << d;
Matrix<int> e;
e = a;
cout << "e : " << endl;
cout << e;
```

for x86 :

```
linjiefang@LAPTOP-99UCO9DE:~/project04$ ./mat 3.txt 1.txt 2.txt
Matrix(int, int)
Matrix(const Matrix &)
b :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
Matrix(int, int, T *)
pointer matrix = 0x7fffbebec4d0
Matrix(const Matrix &)
pointer = 0x7fffc6399650, matrix pointer = 0x7fffbebec4d0, pointTime = 4
c (the submatrix of a) :
row = 3, column = 3
1 2 3
7 8 9
2 3 4
Matrix(int, int, T *)
pointer matrix = 0x7fffbebec524
Matrix(const Matrix &)
pointer = 0x7fffc6399650, matrix pointer = 0x7fffbebec524, pointTime = 4
d (the submatrix of a) :
row = 3, column = 3
7 8 9
9 10 11
7 9 11
Matrix()
the address is 0x7fffbebec4b0, the num of point is 3
e :
row = 6, column = 6
1 2 3 4 5 6
7 8 9 10 11 12
2 3 4 5 6 7
4 5 6 7 8 9
6 7 8 9 10 11
1 3 5 7 9 11
running time : 0.7597 ms
pointer = 0x7fffc63997c0, matrix pointer = 0x7fffbebec4d0, pointTime = 3
pointer = 0x7fffc6399790, matrix pointer = 0x7fffbebec524, pointTime = 3
pointer = 0x7fffc6399760, matrix pointer = 0x7fffbebec4d0, pointTime = 3
pointer = 0x7fffc6399730, matrix pointer = 0x7fffbebec4d0, pointTime = 2
pointer = 0x7fffc6399700, matrix pointer = 0x7fffbebec4d0, pointTime = 1
```

for ARM:



3.3.2.2 same

The results of the program are in line with expectations, and the program has high portability.

4 Reference

https://github.com/ShiqiYu/libfacedetection/issues/255

https://www.cnblogs.com/cxq0017/p/6076856.html

And a lot of blogs on CSDN.