CS205 C/ C++ Programming Project 4: A Class for Matrices

Name: 睢和（SUI He)

SID: 12012929

Requirement

1. Design a class for matrices, and the class should contain the data of a matrix and related information such the number of rows, the number of columns, the number of channels, etc.
2. The class support different data types. It means that the matrix elements can be `unsigned char`, `short`, `int`, `float`, `double`, etc.
3. Do not use memory hard copy if a matrix object is assigned to another. Please carefully handle the memory management to avoid memory leak and to release memory multiple times.
4. Implement some frequently used operators including but not limit to `=`, `==`, `+`, `-`, `*`, etc. Surely the matrix multiplication in Project 3 should be included.
5. Implement region of interest (ROI) to avoid memory hard copy.
6. Test your program on X86 and ARM platforms, and describe the differences.
7. Class `cv::Mat` is a good example for this project. https://docs.opencv.org/master/d3/d63/classcv_1_1Mat.html

Part 1 - Matrix类的构建

支持不同数据类型

为了让矩阵支持`unsigned char`, `short`, `int`, `float`, `double`等数据类型的操作，本次project中我使用了模板类。这样设计的目的主要是使用该类时更加容易操作，并且进行数据修改时可以直接进行修改，不需要先进行指针之间的强制转换。

在构建新的矩阵的时候只需要显示地指定矩阵数据类型，如`Matrix<unsigned char> a`, `Matrix<short> b`, `Matrix<int> c`, `Matrix<float> d`, `Matrix<double> f`， 即可构建不同数据类型的矩阵。

Matrix类的成员变量

在本次project的矩阵类中，设计的成员变量如下：

```
class Matrix
{
private:
    int rows;
    int cols;
    int channel;
    int step;
    T *data;
    T *data_start;
    T *data_end;
    bool isSubMatrix;
    int *refcount;

public:
    //other functions
};
```

rows, cols, channel, data, step

矩阵类中存储了矩阵的信息，包括矩阵的行数，列数，通道数，以及数据。为了实现多通道功能以及连续存储，`data`中数据存储方式为按行存储，对于一行中的每一个元素，依次存储该元素中的不同通道中的值，在一行数据存储完毕后，若该矩阵不是子矩阵，后面紧接着存储下一行数据，依次类推。若该矩阵是其他矩阵的子矩阵，则间隔一定空间存储下一行数据。具体间隔由`step`变量决定，该变量记录了原矩阵的矩阵宽度，因此可以找到子矩阵的下一行元素所在位置。

refcount

矩阵类中有多种操作如复制矩阵，或者取矩阵的roi子矩阵，为了提高程序效率，没有采用hard copy,采用的全部为soft copy。但是，在释放其中一个矩阵的时候，如果直接把存储数据的数组释放了，调用其他矩阵就会产生问题。

因此，使用了指针记录了该指针引用的次数。在矩阵创建的时候对这个指针进行初始化，之后有其他矩阵复制该矩阵或者取roi区域，只需要增加引用次数。在释放矩阵的时候，减少引用次数。并且只在没有矩阵引用该数组的时候，释放存储数据的数组。

将这个变量设置为指针的目的是可以让引用同一矩阵的引用次数同步更新，而不需要对每个矩阵的引用次数单独更新。该变量对于矩阵的内存动态管理具有重要的作用。

data_start, data_end

这两个变量记录了原矩阵的数据起始位置和终止位置，这两个变量主要在`Matrix &adjustROI(int dtop, int dbottom, int dleft, int dright)`方法中使用。该方法用于调整子矩阵在大矩阵的区域，而`data_start`和`data_end`用于帮助确定子矩阵在原矩阵中的位置从而调整子矩阵的大小及位置。该方法会在后续介绍。

成员变量属性：`private`

为了防止使用者随意改变矩阵的成员变量，而在使用过程中产生问题，这矩阵类中将其全部设置为`private`。使用者只能通过相应`get`方法获取成员函数的值，而修改也需要通过相应的方法进行修改。在这些方法中也同时对传入数据进行判断，只有修改要求合法才会进行相应操作，保证了程序不会因为非法操作而崩溃。

Matrix类的函数

Matirx类一共设计了以下函数：

```cpp
template <class T>
class Matrix
{
private:
    //member variables
public:
    Matrix();
    Matrix(const Matrix &m);
    Matrix(int rows, int cols, int channel = 1);
    Matrix(const Matrix &b, int x, int y, int row, int col);
    Matrix &adjustROI(int dtop, int dbottom, int dleft, int dright);
    void readFile(string file_path);
    Matrix subMatrix(int x, int y, int row, int col) const;
    Matrix clone() const;
    T &at(int x, int y, int z = 1);
    const Matrix operator=(const Matrix &b);
    const Matrix operator=(const T b);
    Matrix operator+(const Matrix &b) const;
    Matrix operator+(const T b) const;
    friend Matrix operator+(const T a, const Matrix &b);
    Matrix operator-(const Matrix &b) const;
    Matrix operator-(const T b) const;
    friend Matrix operator-(const T a, const Matrix &b);
    Matrix operator*(const Matrix &b) const;
    Matrix operator*(const T b) const;
    friend Matrix operator*(const T a, const Matrix &b);
    bool operator==(const Matrix &b) const;
    bool operator==(const T b) const;
    friend bool operator==(const T a, const Matrix &b);
    bool operator!=(const Matrix &b) const;
    bool operator!=(const T b) const;
    friend bool operator!=(const T a, const Matrix &b);
    int getRows() const;
    int getCols() const;
    int getChannel() const;
    bool getIsSubMatrix() const;
    int getReferenceCount() const;
    friend ostream &operator<<(ostream &os, const Matrix &matrix);
    friend ofstream &operator<<(ofstream &ofs, const Matrix &matrix);
};
```

Constructor

Matrix()

该构造器为`Matrix`类的默认构造器，生成一个行数、列数、通道数均为0，`data = NULL`的空矩阵，只初始化`refcount`并设置为1。

Matrix(const Matrix &m)

该构造器为`Matrix`的复制构造器，将`m`的成员变量以及数据的指针依次赋值给新矩阵，并让`refcount`增加1。该复制函数为软复制，对其矩阵内容修改也会导致原矩阵值一起改变。

Matrix(int rows, int cols, int channel = 1)

该构造器生成一个对应行数、列数、通道数的矩阵，若没有传入通道数，则通道数默认为1。存储数据的数组共有`rows * cols * channel`个数，并初始化为0。若`rows`、`cols`或`channel`为非正数，则调用默认构造器`Matrix()`返回空矩阵

Matrix(const Matrix &b, int x, int y, int row, int col)

该构造器生成矩阵b的roi区域的矩阵，x和y为新矩阵头在矩阵`b`中的位置(x和y均从1开始)，row和col为roi区域的行数和列数。首先通过判断是否能取给定的roi区域，如果不能，则调用`Matrix()`返回空矩阵。之后计算出新矩阵的矩阵头地址并赋值给`data`，并更新`data_start`和`data_end`。并将引用次数+1。

void release()

该方法用于矩阵的释放。首先将引用次数-1，若引用次数等于0，即表示没有矩阵继续引用这一块区域，释放该矩阵的**原矩阵**的数组头（即为`data_start`)以及`refcount`。否则，不对数组进行操作。最后再将矩阵内各项成员变量设置为0，并将`data`,`data_start`,`data_end`设置为`NULL`,防止用户还能通过这些指针对数据进行操作。

~Matrix()

调用`realease()`方法

Matrix &adjustROI(int dtop, int dbottom, int dleft, int dright)

该方法用于调整子矩阵在原矩阵的roi区域，四个参数分别表示向四个方向移动的距离。例如`dtop = 2`表示将矩阵的上边界向上移动2，`dtop = -1`表示将矩阵的上边界向下移动1，`dleft = 3`表示将矩阵的左边界向左移动1，`dleft = -4`表示将矩阵的左边界向右移动4。其他全部同理。

该方法首先需要根据`data`，`data_start`，`data_end`和`step`确定子矩阵在原矩阵的位置，并推算出原矩阵的行数。然后计算出矩阵四个边界的位置。

规定调整的数量不能超过原矩阵的范围，若超出原矩阵范围，则边界即为原矩阵的边界。但若出现上边界在下边界下面或左边界在有边界的右边，则不进行调整。

最后计算出矩阵头的位置并更新`data`，以及行数和列数，并根据情况更新`isSubMatrix`。

整个调整的过程均没有深复制的操作。

Matrix subMatrix(int x, int y, int row, int col) const

作用同构造器`Matrix(const Matrix &b, int x, int y, int row, int col)`。

该方法`return Matrix(*this, x, y, row, col);`

重载操作符

在该矩阵类中，重载了`+`,`-`,`*`,`=`,`==`,`!=`操作符。在该矩阵类中，设计时，只允许同种数据类型的矩阵的运算，而没有添加自动数据类型转换。理由如下：

1. 矩阵数据类型转换较为繁琐
2. 矩阵类型不方便自动转换，例如`int`和`float`类型矩阵相互转换，如果将`int`转为`float`，在`int`值较大时，转换为`float`会有精度损失，同样，`float`转换为`int`也会有精度损失。具体转换应该根据使用者需求而定。于是设计只能同种数据类型进行运算，并且使用者需要注意选择合适的数据类型防止超过数据范围。

重载操作符返回类型的考虑

对于操作符`+`，`-`，`*`，在设计的时候考虑过返回的数据类型应该是`Matrix`还是`Matrix&`.

如果返回的是`Matrix&`，即矩阵的引用。若在方法内通过`Matrix<int> result`方式定义的返回矩阵，该矩阵是一个临时变量，出了方法就会被销毁，不可行。但如果通过`Matrix<int> * result = new Matrix`生成矩阵，返回时返回`*result`,能够实现。但是如果一个式子里有多个运算，如`Matrix<int> a = b * c + d`,则会产生中间变量`b * c`,并且我们无法访问到这个中间变量，也无法对其进行内存管理。

因此，返回类型应该要是`Matrix`，在返回的时候先调用复制构造函数，再调用析构函数销毁临时变量。这样，即可解决中间变量无法处理的问题。

注：以下所有操作符均可作用于子矩阵。

const Matrix operator=(const Matrix &b)

该方法先调用`release()`将自己释放，然后将b的成员变量依次赋值给当前矩阵，并且将引用变量增加1。为了能够实现连等例如`a = b = c`,方法返回自己的引用。

const Matrix &operator=(const T b)

将该矩阵各元素各通道赋值为b。

Matrix operator+(const Matrix &b) const

若两个矩阵规模不一样，则调用`Matrix()`返回空矩阵。

该方法返回当前矩阵与矩阵`b`两个矩阵相同位置的元素的各通道分别相加的矩阵。

Matrix operator+(const T b) const

该方法返回当前矩阵内各元素的各通道加上`b`的矩阵。

friend Matrix operator+(const T a, const Matrix &b)

调用`b + a`。

Matrix operator-(const Matrix &b) const

若两个矩阵规模不一样，则调用`Matrix()`返回空矩阵。

该方法返回当前矩阵与矩阵`b`两个矩阵相同位置的元素的各通道分别相减的矩阵。

Matrix operator+(const T b) const

该方法返回当前矩阵内各元素的各通道减去`b`的矩阵。

friend Matrix operator+(const T a, const Matrix &b)

该方法返回当前矩阵内`a`分别减各元素的各通道的矩阵。

Matrix operator*(const Matrix &b) const

若当前矩阵的列数不等于`b`的行数或通道数不同，则不能相乘，调用`Matrix()`返回空矩阵。

该方法返回的是矩阵各通道矩阵分别相乘组合起来的结果，即this的第一通道 * b的第一通道，this的第二通道 * b的第二通道...再组合形成的矩阵。

Matrix operator*(const T b) const

该方法返回当前矩阵内各元素的各通道乘`b`的矩阵。

friend Matrix operator*(const T a, const Matrix &b)

调用`b * a`。

bool operator==(const Matrix &b) const

该方法返回两个矩阵的各元素的各通道是否分别相同。

bool operator==(const T b) const

该方法返回当前矩阵的各元素的各通道是否均等于`b`。

friend bool operator==(const T a, const Matrix &b)

调用`b == a`。

bool operator!=(const Matrix &b) const

该方法返回两个矩阵是否存在元素不同。

bool operator!=(const T b) const

该方法返回当前矩阵是否存在元素不等于`b`

friend bool operator!=(const T a, const Matrix &b)

调用 `b != a`。

T &at(int x, int y, int z = 1)

返回矩阵第x行，第y列，第z通道数据的引用（行数和列数均为从1算起）。若不传入z，则默认第一通道。若传入的位置不存在，则返回矩阵头。

void readFile(string file_path)

该方法用于从文件中读取数据存储到矩阵。

friend ofstream &operator<<(ofstream &ofs, const Matrix &matrix)

将矩阵数据插入到输出流中打印出来。

Matrix clone() const

如果对于数据需要进行深拷贝，可以使用`clone()`函数，该函数使用`memcpy()`进行快速拷贝。如果要拷贝的矩阵不是子矩阵（存储是连续存储的），即可一次性快速拷贝完全。若该矩阵是其他矩阵的子矩阵（存储不连续），只能每行分别调用`memcpy()`进行拷贝。

模板类声明与实现分开

按照常规习惯，类的声明放在`.hpp`文件中，具体实现放在`.cpp`文件中。而本次类是模板类，我依旧尝试分开方法的声明与定义，但是发现会产生链接错误。

经过了解，C++中每一个对象所占用的空间大小，是在编译的时候就确定的，在模板类没有真正的使用之前，编译器无法知道模板类中使用模板类型的对象所占用空间的大小。只有模板真正使用的时候，编译器才知道模板套用的是什么类型，应该分配多少空间。而编译器和连接器某处有一机制会去掉指定模板的多重定义，导致在链接的时候出错。

但是我还是希望将声明和实现分开，以更好实现类的封装。后来我发现可以在在头文件中显示地声明实例化对象，并在`.cpp`文件中实例化具体的对象。

```
template class Matrix<unsigned char>;
template class Matrix<short>;
template class Matrix<int>;
template class Matrix<float>;
template class Matrix<double>;
```

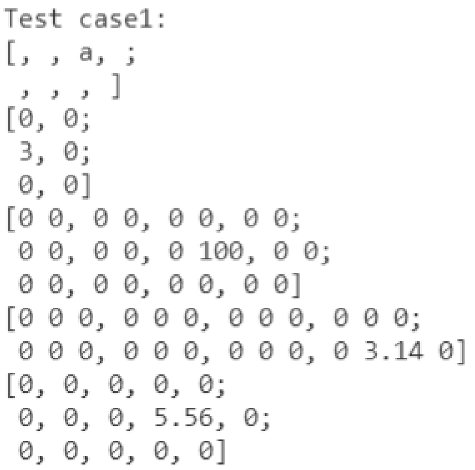
做了以上操作后，成功将类的声明与实现分开。

Part 2 - Result & Verification

Test case1: 构造不同数据类型矩阵

```
    //Test case1:
    {
        cout << "Test case1:" << endl;
        Matrix<unsigned char> A(2,4);
        Matrix<short> B(3, 2);
        Matrix<int> C(3, 4, 2);
        Matrix<float> D(2, 4, 3);
        Matrix<double> E(3, 5, 1);
        A.at(1, 3) = 'a';
        B.at(2, 1) = 3;
        C.at(2, 3, 2) = 100;
        D.at(2, 4, 2) = 3.14;
        E.at(2, 4, 1) = 5.56;
        cout << A << B << C << D << E;
    }
```

测试结果：



```
Test case1:
[, , a, ;
 , , , ]
[0, 0;
 3, 0;
 0, 0]
[0 0, 0 0, 0 0, 0 0;
 0 0, 0 0, 0 100, 0 0;
 0 0, 0 0, 0 0, 0 0]
[0 0 0, 0 0 0, 0 0 0, 0 0 0;
 0 0 0, 0 0 0, 0 0 0, 0 3.14 0]
[0, 0, 0, 0, 0;
 0, 0, 0, 5.56, 0;
 0, 0, 0, 0, 0]
```

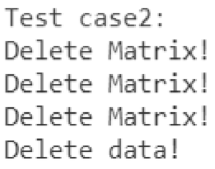
Test case2: 测试析构函数

测试时在释放数组处加上了`cout << "Delete data!" << endl;`，析构函数处加上了`cout << "Delete Matrix!" << endl;`以便观察数组什么时候释放的。

```
    //Test case2:
    {
        Matrix<int> A(3, 5);
        Matrix<int> B(A, 1, 2, 2, 2);
        Matrix<int> C(A, 1, 1, 3, 2);
    }
```

B和C均为A的子矩阵，和A共用数据域。

测试结果：



```
Test case2:
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete data!
```

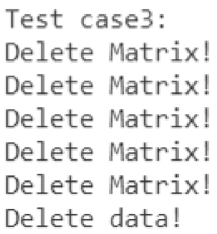
在离开大括号时分别会释放这些矩阵，可以看出一共调用了三次析构函数，但只有最后一次释放了数组。

Test case3: 先删除大矩阵，再删除其子矩阵

```
    //Test case3:
    {
        cout << "Test case2:" << endl;
        Matrix<int> B;
        Matrix<int> C;
        {

            Matrix<int> A(3, 5);
            B = A.subMatrix(1, 2, 2, 2);
            C = A.subMatrix(1, 1, 3, 2);
        }
    }
```

测试结果：



```
Test case3:
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete data!
```

前两个"Delete Matrix!"为在用`=`赋值时先对B和C原先数据进行释放时输出的，之后A离开作用域A先被删除，但是还没有释放数组数据，等引用他的B和C均离开作用域才释放数组数据。

注：为方便进行测试，在接下来的测试中矩阵类型均为`int`，并预先定义如下两个矩阵A和B，并在接下来测试样例中通用

```
    Matrix<int> A(6, 7, 3);
    Matrix<int> B(5, 8, 3);
    for (int i = 1; i <= A.getRows(); ++i)
        for (int j = 1; j <= A.getCols(); ++j)
            for (int k = 1; k <= A.getChannel(); ++k)
                A.at(i, j, k) = 3 * i - j + 5 * k;
    for (int i = 1; i <= B.getRows(); ++i)
        for (int j = 1; j <= B.getCols(); ++j)
            for (int k = 1; k <= B.getChannel(); ++k)
                B.at(i, j, k) = 2 * i + j + 3 * k;
```

> A =
>
> [7 12 17, 6 11 16, 5 10 15, 4 9 14, 3 8 13, 2 7 12, 1 6 11;
> 10 15 20, 9 14 19, 8 13 18, 7 12 17, 6 11 16, 5 10 15, 4 9 14;
> 13 18 23, 12 17 22, 11 16 21, 10 15 20, 9 14 19, 8 13 18, 7 12 17;
> 16 21 26, 15 20 25, 14 19 24, 13 18 23, 12 17 22, 11 16 21, 10 15 20;
> 19 24 29, 18 23 28, 17 22 27, 16 21 26, 15 20 25, 14 19 24, 13 18 23;
> 22 27 32, 21 26 31, 20 25 30, 19 24 29, 18 23 28, 17 22 27, 16 21 26]

> B =
>
> [6 9 12, 7 10 13, 8 11 14, 9 12 15, 10 13 16, 11 14 17, 12 15 18, 13 16 19;
> 8 11 14, 9 12 15, 10 13 16, 11 14 17, 12 15 18, 13 16 19, 14 17 20, 15 18 21;
> 10 13 16, 11 14 17, 12 15 18, 13 16 19, 14 17 20, 15 18 21, 16 19 22, 17 20 23;
> 12 15 18, 13 16 19, 14 17 20, 15 18 21, 16 19 22, 17 20 23, 18 21 24, 19 22 25;
> 14 17 20, 15 18 21, 16 19 22, 17 20 23, 18 21 24, 19 22 25, 20 23 26, 21 24 27]

不同元素间用逗号分隔，如A中`7 12 17,`表示A的第一行第一列的三个通道的元素分别为7, 12, 17。

Test case4: ROI取子矩阵

```
    //Test case4:
    {
        cout << "Test case4:" << endl;
        Matrix<int> C(A, 2, 4, 3, 2);
        Matrix<int> D = B.subMatrix(3, 5, 3, 3);
        cout << C << D;
    }
```

测试结果：

```
Test case4:
[7 12 17, 6 11 16;
 10 15 20, 9 14 19;
 13 18 23, 12 17 22]
[14 17 20, 15 18 21, 16 19 22;
 16 19 22, 17 20 23, 18 21 24;
 18 21 24, 19 22 25, 20 23 26]
```

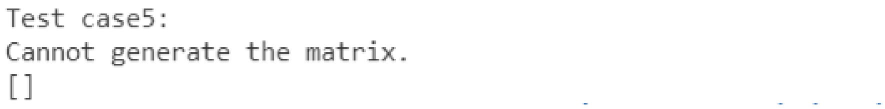
C取出了以A的第2行，第4列为矩阵头，行数为3，列数为2的子矩阵

D取出了以B的第3行，第5列为矩阵头，行数为3，列数为3的子矩阵

Test case5: ROI取的矩阵超过原数组范围

```cpp
    //Test case5:
    {
        cout << "Test case5:" << endl;
        Matrix<int> C(A, 2, 5, 3, 4);
        cout << C;
    }
```

测试结果:

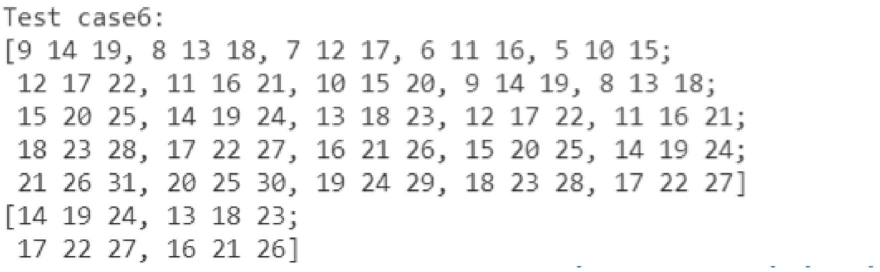


```
Test case5:
Cannot generate the matrix.
[]
```

Test case6: 在子矩阵中继续取子矩阵

```cpp
    //Test case6:
    {
        cout << "Test case6:" << endl;
        Matrix<int> C(A, 2, 2, 5, 5);
        cout << C;
        Matrix<int> D;
        D = C.subMatrix(3, 2, 2, 2);
        cout << D;
    }
```

测试结果:

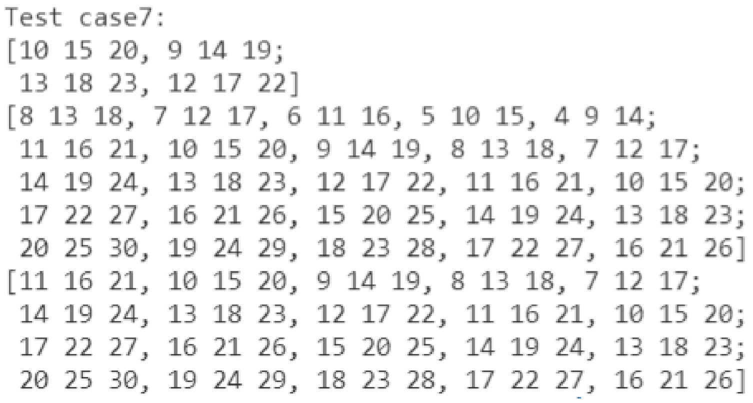


```
Test case6:
[9 14 19, 8 13 18, 7 12 17, 6 11 16, 5 10 15;
 12 17 22, 11 16 21, 10 15 20, 9 14 19, 8 13 18;
 15 20 25, 14 19 24, 13 18 23, 12 17 22, 11 16 21;
 18 23 28, 17 22 27, 16 21 26, 15 20 25, 14 19 24;
 21 26 31, 20 25 30, 19 24 29, 18 23 28, 17 22 27]
[14 19 24, 13 18 23;
 17 22 27, 16 21 26]
```

Test case7: 调整子矩阵的ROI区域

```cpp
    //Test case7:
    {
        cout << "Test case7:" << endl;
        Matrix<int> C(A, 3, 4, 2, 2);
        cout << C;
        C.adjustROI(1, 2, 1, 4);
        cout << C;
        C.adjustROI(-1, 3, 0, 0);
        cout << C;
    }
```

测试结果:



```
Test case7:
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
[8 13 18, 7 12 17, 6 11 16, 5 10 15, 4 9 14;
 11 16 21, 10 15 20, 9 14 19, 8 13 18, 7 12 17;
 14 19 24, 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 17 22 27, 16 21 26, 15 20 25, 14 19 24, 13 18 23;
 20 25 30, 19 24 29, 18 23 28, 17 22 27, 16 21 26]
[11 16 21, 10 15 20, 9 14 19, 8 13 18, 7 12 17;
 14 19 24, 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 17 22 27, 16 21 26, 15 20 25, 14 19 24, 13 18 23;
 20 25 30, 19 24 29, 18 23 28, 17 22 27, 16 21 26]
```

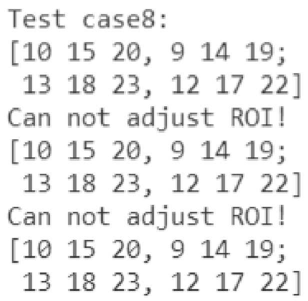
第一次调整分别让上,下,左,右边界分别增长1,2,1,4.

第二次调整让上边界往右移1,下边界增3,但增长超过了原矩阵下边界,固定下边界为原矩阵下边界.

Test case8: 调整ROI时上边界在下边界下面或左边界在右边界右边导致调整失败

```
    //Test case8:
    {
        cout << "Test case8:" << endl;
        Matrix<int> C(A, 3, 4, 2, 2);
        cout << C;
        C.adjustROI(-2, -1, 0, 0);
        cout << C;
        C.adjustROI(0, 0, -1, -3);
        cout << C;
    }
```

测试结果:



```
Test case8:
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
Can not adjust ROI!
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
Can not adjust ROI!
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
```

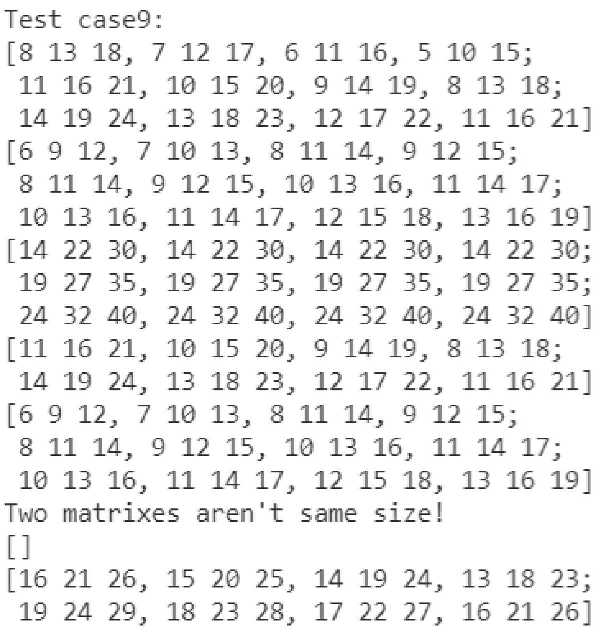
Test case9: 矩阵加法

```
    //Test case9:
    {
        cout << "Test case9:" << endl;
        Matrix<int> C(A, 2, 3, 3, 4);
        Matrix<int> D(B, 1, 1, 3, 4);
        cout << C << D;
        Matrix<int> E = C + D;
        cout << E;
        C.adjustROI(-1, 0, 0, 0);
        cout << C << D;
        cout << C + D;

        Matrix<int> F = 5 + C;
```

```cpp
        cout << F;
    }
```

测试结果:



```
Test case9:
[8 13 18, 7 12 17, 6 11 16, 5 10 15;
 11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
[14 22 30, 14 22 30, 14 22 30, 14 22 30;
 19 27 35, 19 27 35, 19 27 35, 19 27 35;
 24 32 40, 24 32 40, 24 32 40, 24 32 40]
[11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
Two matrixes aren't same size!
[]
[16 21 26, 15 20 25, 14 19 24, 13 18 23;
 19 24 29, 18 23 28, 17 22 27, 16 21 26]
```

第一次可加正常相加.

第二次调整C的规模后无法相加.

第三次为矩阵与常数的加法.

Test case10: 矩阵减法

```cpp
    //Test case10:
    {
        cout << "Test case10:" << endl;
        Matrix<int> C(A, 2, 3, 3, 4);
        Matrix<int> D(B, 1, 1, 3, 4);
        cout << C << D;
        Matrix<int> E = C - D;
        cout << E;
        C.adjustROI(-1, 0, 0, 0);
        cout << C << D;
        cout << C - D;

        Matrix<int> F = 30 - C;
        cout << F;
    }
```

测试结果:

```
Test case10:
[8 13 18, 7 12 17, 6 11 16, 5 10 15;
 11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
[2 4 6, 0 2 4, -2 0 2, -4 -2 0;
 3 5 7, 1 3 5, -1 1 3, -3 -1 1;
 4 6 8, 2 4 6, 0 2 4, -2 0 2]
[11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
Two matrixes aren't same size!
[]
[19 14 9, 20 15 10, 21 16 11, 22 17 12;
 16 11 6, 17 12 7, 18 13 8, 19 14 9]
```

Test case11: 矩阵乘法

```
    //Test case11:
    {
        cout << "Test case11:" << endl;
        Matrix<int> C(A, 2, 1, 3, 4);
        Matrix<int> D(B, 2, 3, 4, 5);
        cout << C << D;
        cout << C * D;
        D.adjustROI(0, -1, 0, 0);
        cout << C * D;
        cout << C * 2;
    }
```

测试结果:

```
Test case11:
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[10 13 16, 11 14 17, 12 15 18, 13 16 19, 14 17 20;
 12 15 18, 13 16 19, 14 17 20, 15 18 21, 16 19 22;
 14 17 20, 15 18 21, 16 19 22, 17 20 23, 18 21 24;
 16 19 22, 17 20 23, 18 21 24, 19 22 25, 20 23 26]
[432 854 1396, 466 908 1470, 500 962 1544, 534 1016 1618, 568 1070 1692;
 588 1046 1624, 634 1112 1710, 680 1178 1796, 726 1244 1882, 772 1310 1968;
 744 1238 1852, 802 1316 1950, 860 1394 2048, 918 1472 2146, 976 1550 2244]
Can not multiple these two matrices.
[]
[20 30 40, 18 28 38, 16 26 36, 14 24 34;
 26 36 46, 24 34 44, 22 32 42, 20 30 40;
 32 42 52, 30 40 50, 28 38 48, 26 36 46]
```

Test case12: `==`和`!=`

```
    //Test case12:
    {
        cout << "Test case12:" << endl;
        Matrix<int> C(2, 3);
        C = 3;
        Matrix<int> D(2, 3);
        D = 3;
        cout << (C == D) << endl;
        cout << (C != D) << endl;
        cout << (A == B) << endl;
        cout << (A != B) << endl;
    }
```

测试结果:

```
Test case12:
1
0
0
1
```

Test case13: `clone()`

```
    //Test case13:
    {
        cout << "Test case13:" << endl;
        Matrix<int> C(A, 2, 1, 3, 4);
        cout << C;
        Matrix<int> D = C.clone();
        cout << D;
        C.at(1, 1, 1) = 0;
        cout << C;
        cout << D;
    }
```

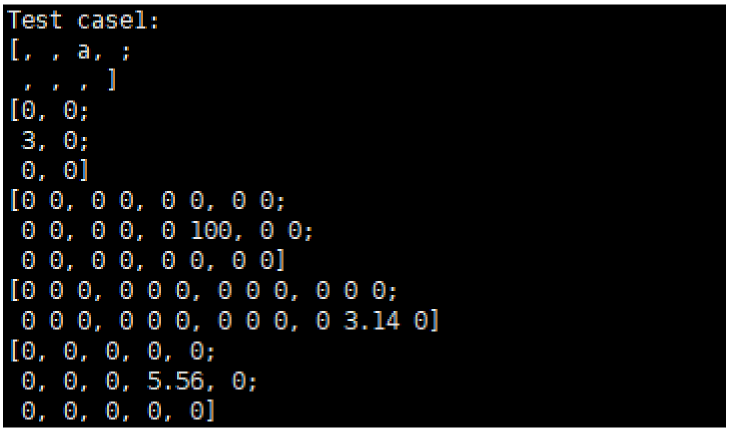
测试结果:

```
Test case13:
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[0 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
```

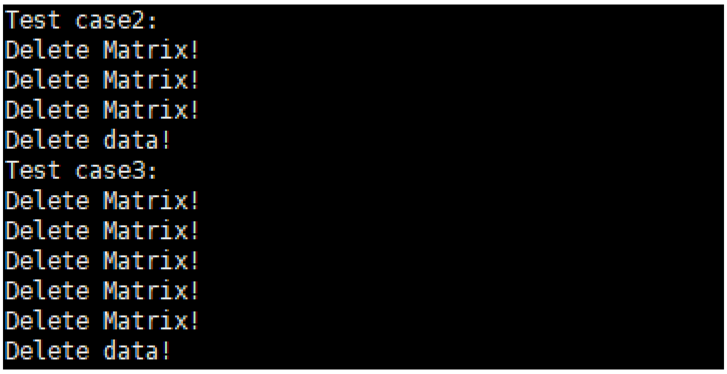
Part 3 - Test Program on ARM platform

Part 2中测试数据

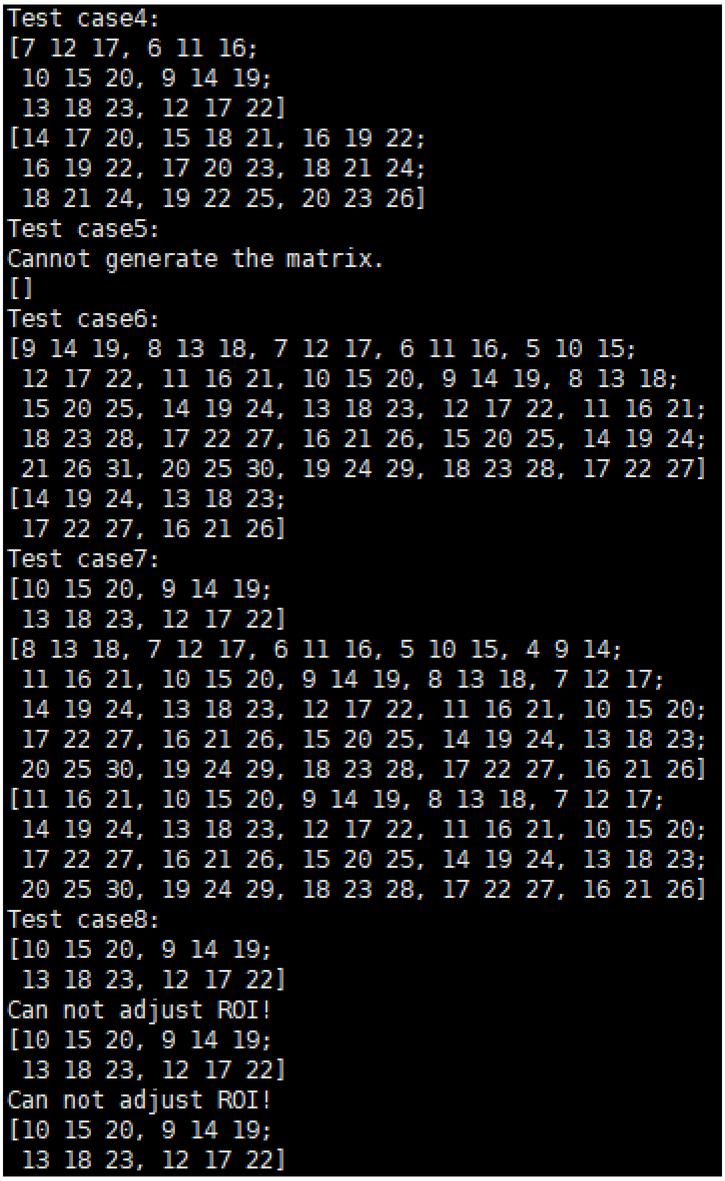
测试结果均一致



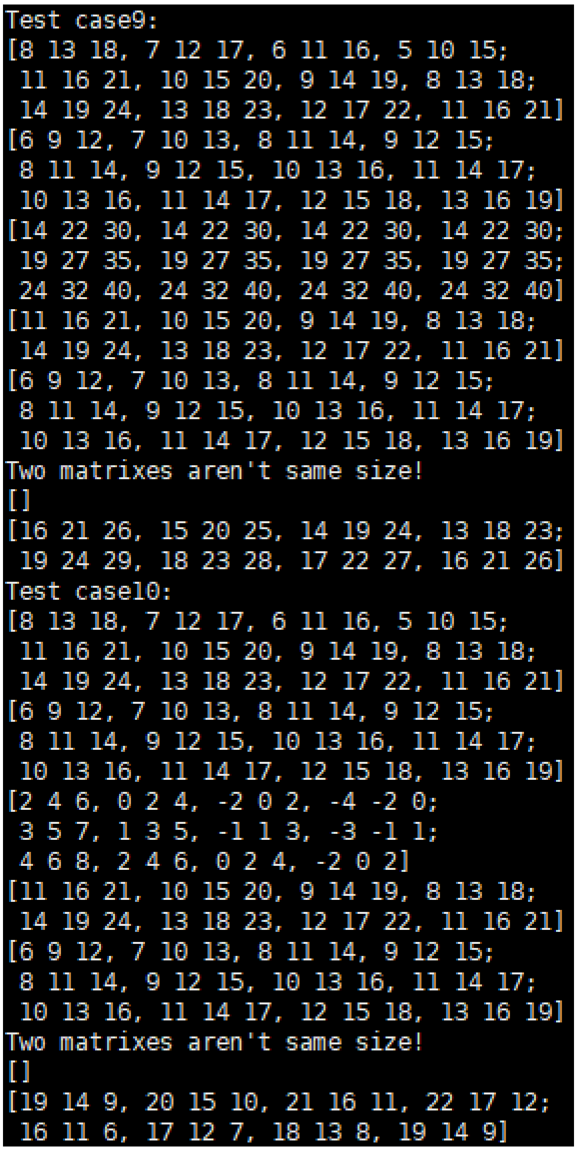
```
Test case1:
[, , a, ;
 , , , ]
[0, 0;
 3, 0;
 0, 0]
[0 0, 0 0, 0 0, 0 0;
 0 0, 0 0, 0 100, 0 0;
 0 0, 0 0, 0 0, 0 0]
[0 0 0, 0 0 0, 0 0 0, 0 0 0;
 0 0 0, 0 0 0, 0 0 0, 0 3.14 0]
[0, 0, 0, 0, 0;
 0, 0, 0, 5.56, 0;
 0, 0, 0, 0, 0]
```



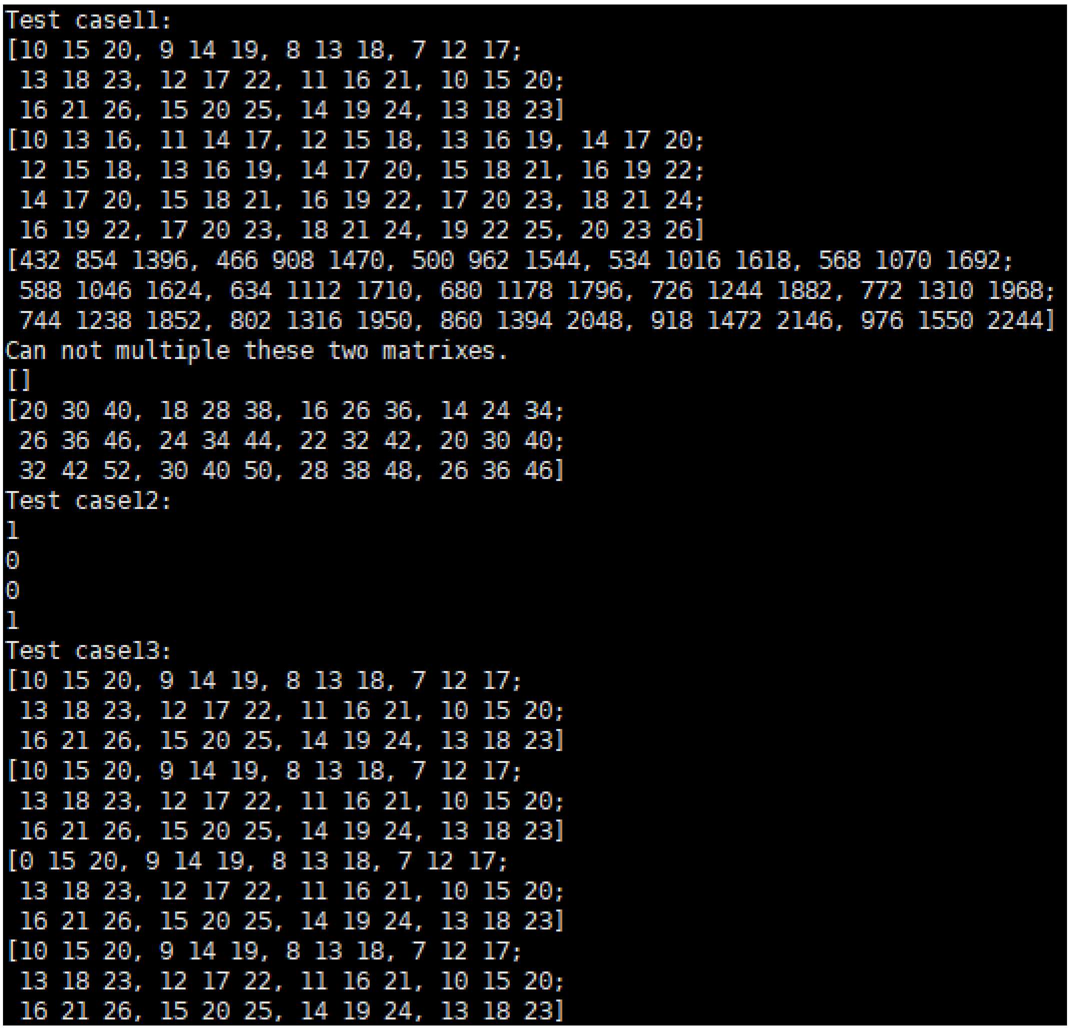
```
Test case2:
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete data!
Test case3:
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete Matrix!
Delete data!
```



```
Test case4:
[7 12 17, 6 11 16;
 10 15 20, 9 14 19;
 13 18 23, 12 17 22]
[14 17 20, 15 18 21, 16 19 22;
 16 19 22, 17 20 23, 18 21 24;
 18 21 24, 19 22 25, 20 23 26]
Test case5:
Cannot generate the matrix.
[]
Test case6:
[9 14 19, 8 13 18, 7 12 17, 6 11 16, 5 10 15;
 12 17 22, 11 16 21, 10 15 20, 9 14 19, 8 13 18;
 15 20 25, 14 19 24, 13 18 23, 12 17 22, 11 16 21;
 18 23 28, 17 22 27, 16 21 26, 15 20 25, 14 19 24;
 21 26 31, 20 25 30, 19 24 29, 18 23 28, 17 22 27]
[14 19 24, 13 18 23;
 17 22 27, 16 21 26]
Test case7:
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
[8 13 18, 7 12 17, 6 11 16, 5 10 15, 4 9 14;
 11 16 21, 10 15 20, 9 14 19, 8 13 18, 7 12 17;
 14 19 24, 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 17 22 27, 16 21 26, 15 20 25, 14 19 24, 13 18 23;
 20 25 30, 19 24 29, 18 23 28, 17 22 27, 16 21 26]
[11 16 21, 10 15 20, 9 14 19, 8 13 18, 7 12 17;
 14 19 24, 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 17 22 27, 16 21 26, 15 20 25, 14 19 24, 13 18 23;
 20 25 30, 19 24 29, 18 23 28, 17 22 27, 16 21 26]
Test case8:
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
Can not adjust ROI!
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
Can not adjust ROI!
[10 15 20, 9 14 19;
 13 18 23, 12 17 22]
```



```
Test case9:
[8 13 18, 7 12 17, 6 11 16, 5 10 15;
 11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
[14 22 30, 14 22 30, 14 22 30, 14 22 30;
 19 27 35, 19 27 35, 19 27 35, 19 27 35;
 24 32 40, 24 32 40, 24 32 40, 24 32 40]
[11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
Two matrixes aren't same size!
[]
[16 21 26, 15 20 25, 14 19 24, 13 18 23;
 19 24 29, 18 23 28, 17 22 27, 16 21 26]
Test case10:
[8 13 18, 7 12 17, 6 11 16, 5 10 15;
 11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
[2 4 6, 0 2 4, -2 0 2, -4 -2 0;
 3 5 7, 1 3 5, -1 1 3, -3 -1 1;
 4 6 8, 2 4 6, 0 2 4, -2 0 2]
[11 16 21, 10 15 20, 9 14 19, 8 13 18;
 14 19 24, 13 18 23, 12 17 22, 11 16 21]
[6 9 12, 7 10 13, 8 11 14, 9 12 15;
 8 11 14, 9 12 15, 10 13 16, 11 14 17;
 10 13 16, 11 14 17, 12 15 18, 13 16 19]
Two matrixes aren't same size!
[]
[19 14 9, 20 15 10, 21 16 11, 22 17 12;
 16 11 6, 17 12 7, 18 13 8, 19 14 9]
```



```
Test case11:
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[10 13 16, 11 14 17, 12 15 18, 13 16 19, 14 17 20;
 12 15 18, 13 16 19, 14 17 20, 15 18 21, 16 19 22;
 14 17 20, 15 18 21, 16 19 22, 17 20 23, 18 21 24;
 16 19 22, 17 20 23, 18 21 24, 19 22 25, 20 23 26]
[432 854 1396, 466 908 1470, 500 962 1544, 534 1016 1618, 568 1070 1692;
 588 1046 1624, 634 1112 1710, 680 1178 1796, 726 1244 1882, 772 1310 1968;
 744 1238 1852, 802 1316 1950, 860 1394 2048, 918 1472 2146, 976 1550 2244]
Can not multiple these two matrixes.
[]
[20 30 40, 18 28 38, 16 26 36, 14 24 34;
 26 36 46, 24 34 44, 22 32 42, 20 30 40;
 32 42 52, 30 40 50, 28 38 48, 26 36 46]
Test case12:
1
0
0
1
Test case13:
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[0 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
[10 15 20, 9 14 19, 8 13 18, 7 12 17;
 13 18 23, 12 17 22, 11 16 21, 10 15 20;
 16 21 26, 15 20 25, 14 19 24, 13 18 23]
```

大矩阵乘法

本次测试的为`float`类型的1024 * 2048三通道矩阵和2048 * 1024三通道矩阵的矩阵乘法。乘法采用的是对每个通道单独采用`ikj`循环计算。

	X86	ARM
关闭编译优化	21659ms	39110ms
开启编译优化	6654ms	11119ms
提升倍数	3.2	3.5

本程序运算是单线程计算，通过计算程序运行时间，ARM平台在计算速度上稍慢于X86平台，但开启编译优化后提升率略微高于X86平台。

但据了解，X86的平台在性能方面一般比ARM平台要快得多、强地多。而ARM的优势不在于性能强大而在于效率，ARM采用RISC流水线指令集，在一些任务相对固定的场所能发挥其优势。

另外，ARM在处理多线程的多核问题上具有很大的优势，但因为时间原因，没有实现多线程乘法，可以在未来使用多线程实现再进行测试。

Part 4 - Source Code

matrix.hpp

```
#pragma once
#pragma GCC optimize(3, "Ofast", "inline")
#include <iostream>
#include <string>
#include <string.h>
#include <fstream>
using namespace std;

template <class T>
class Matrix
{
private:
    int rows;
    int cols;
    int step;
    int channel;
    T *data;
    T *data_start;
    T *data_end;
    bool isSubMatrix;
    int *refcount;

public:
    Matrix();
    Matrix(const Matrix &m);
    Matrix(int rows, int cols, int channel = 1);
    Matrix(const Matrix &b, int x, int y, int row, int col);
    Matrix &adjustROI(int dtop, int dbottom, int dleft, int dright);
    void readFile(string file_path);
    T &at(int x, int y, int z = 1);
    const Matrix &operator=(const Matrix &b);
    const Matrix &operator=(const T b);
    Matrix operator+(const Matrix &b) const;
    Matrix operator+(const T b) const;
    friend Matrix operator+(const T a, const Matrix &b)
    {
        return b + a;
    }

    Matrix operator-(const Matrix &b) const;
    Matrix operator-(const T b) const;
    friend Matrix operator-(const T a, const Matrix &b)
    {
        if (b.data == NULL)
        {
            cerr << "The matrix is empty!" << endl;
            return Matrix();
        }
        Matrix result(b.rows, b.cols, b.channel);
        if (!b.isSubMatrix)
        {
            for (int i = 0; i < b.rows * b.cols * b.channel; ++i)
                result.data[i] = a - b.data[i];
            return result;
        }
```

```cpp
        T *ptr = result.data;
        T *ptrb = b.data;
        for (int i = 0; i < b.rows; ++i)
        {
            for (int j = 0; j < b.cols * b.channel; ++j)
            {
                *ptr = a - *ptrb;
                ++ptr;
                ++ptrb;
            }
            ptrb += (b.step - b.cols) * b.channel;
        }
        return result;
    }

    Matrix operator*(const Matrix &b) const;
    Matrix operator*(const T b) const;
    friend Matrix operator*(const T a, const Matrix &b)
    {
        return b * a;
    }

    bool operator==(const Matrix &b) const;
    bool operator==(const T b) const;
    friend bool operator==(const T a, const Matrix &b)
    {
        return b == a;
    }

    bool operator!=(const Matrix &b) const;
    bool operator!=(const T b) const;
    friend bool operator!=(const T a, const Matrix &b)
    {
        return b != a;
    }
    Matrix clone() const;
    int getRows() const;
    int getCols() const;
    int getChannel() const;
    bool getIsSubMatrix() const;
    int getReferenceCount() const;
    Matrix subMatrix(int x, int y, int row, int col) const;
    friend ostream &operator<<(ostream &os, const Matrix &matrix)
    {
        T *ptr = matrix.data;
        os << "[";
        for (int i = 1; i <= matrix.rows; ++i)
        {
            if (i != 1)
                os << " ";
            for (int j = 1; j <= matrix.cols; ++j)
            {
                for (int k = 1; k <= matrix.channel; ++k)
                {
                    if (j == matrix.cols && k == matrix.channel)
                    {
                        if (i != matrix.rows)
                        {
                            os << *ptr << ";" << endl;
                            ptr += (matrix.step - matrix.cols) * matrix.channel + 1;
                        }
                        else
                            os << *ptr;
                    }
                    else
                    {
                        if (k == matrix.channel)
                            os << *ptr << ", ";
                        else
                            os << *ptr << " ";
                        ++ptr;
                    }
                }
            }
        }
        os << "]" << endl;
        return os;
    }
    friend ofstream &operator<<(ofstream &ofs, const Matrix &matrix)
    {
        T *ptr = matrix.data;
        for (int i = 1; i <= matrix.rows; ++i)
        {
            for (int j = 1; j <= matrix.cols; ++j)
            {
                for (int k = 1; k <= matrix.channel; ++k)
                {
                    if (j == matrix.cols && k == matrix.channel)
                    {
                        if (i != matrix.rows)
                        {
                            ofs << *ptr << ";" << endl;
                            ptr += (matrix.step - matrix.cols) * matrix.channel + 1;
                        }
                        else
                            ofs << *ptr;
                    }
                    else
                    {
                        if (k == matrix.channel)
                            ofs << *ptr << ", ";
                        else
                            ofs << *ptr << " ";
                        ++ptr;
                    }
                }
            }
        }
        return ofs;
    }
    void release();
    ~Matrix();
};

```

```
extern template class Matrix<unsigned char>;
extern template class Matrix<short>;
extern template class Matrix<int>;
extern template class Matrix<float>;
extern template class Matrix<double>;
```

> matrix.cpp

```
#include "matrix.hpp"

template <class T>
Matrix<T>::Matrix()
{
    rows = cols = step = channel = 0;
    data = data_start = data_end = NULL;
    refcount = new int[1];
    refcount[0] = 1;
    isSubMatrix = false;
}

template <class T>
Matrix<T>::Matrix(int rows, int cols, int channel)
{
    if (rows <= 0 || cols <= 0 || channel <= 0)
    {
        rows = cols = step = channel = 0;
        data = data_start = data_end = NULL;
        cerr << "Rows and cols must greater than 0." << endl;
    }
    else
    {
        this->rows = rows;
        this->cols = cols;
        this->channel = channel;
        data = new T[rows * cols * channel]();
        step = cols;
        data_start = data;
        data_end = data + rows * cols * channel;
    }
    refcount = new int[1];
    refcount[0] = 1;
    isSubMatrix = false;
}

template <class T>
Matrix<T>::Matrix(const Matrix<T> &m)
{
    rows = m.rows;
    cols = m.cols;
    data = m.data;
    channel = m.channel;
    step = m.step;
    data_start = m.data_start;
    data_end = m.data_end;
    refcount = m.refcount;
    ++*refcount;
    isSubMatrix = m.isSubMatrix;
}

template <class T>
Matrix<T>::Matrix(const Matrix<T> &b, int x, int y, int rows, int cols)
{
    if (x <= 0 || y <= 0 || rows <= 0 || cols <= 0 || x + rows - 1 > b.rows || y + cols - 1 > b.cols)
    {
        cerr << "Cannot generate the matrix." << endl;
        new (this) Matrix();
    }
    else
    {
        this->rows = rows;
        this->cols = cols;
        channel = b.channel;
        data = b.data + ((x - 1) * b.step + y - 1) * channel;
        step = b.step;
        data_start = b.data_start;
        data_end = b.data_end;
        refcount = b.refcount;
        ++*refcount;
        if (rows == b.rows && cols == b.cols)
            isSubMatrix = false;
        else
            isSubMatrix = true;
    }
}

template <class T>
void Matrix<T>::release()
{
    --*refcount;
    if (*refcount == 0)
    {
        if (data_start != NULL)
        {
            delete[] data_start;
            // cout << "Delete data!" << endl;
        }
        delete[] refcount;
    }
    rows = cols = step = channel = 0;
    data = data_start = data_end = NULL;
    refcount = NULL;
}

template <class T>
Matrix<T>::~Matrix()
{
    // cout << "Delete Matrix!" << endl;
    release();
}

template <class T>
T &Matrix<T>::at(int x, int y, int z)
```

```
{
    if (x < 1 || x > rows || y < 1 || y > cols || z < 1 || z > channel)
        return *data;
    return *(data + ((x - 1) * step + y - 1) * channel + z - 1);
}

template <class T>
Matrix<T> Matrix<T>::operator+(const Matrix<T> &b) const
{
    if (rows != b.rows || cols != b.cols || channel != b.channel)
    {
        cerr << "Two matrixes aren't same size!" << endl;
        return Matrix();
    }
    if (data == NULL || b.data == NULL)
    {
        cerr << "The matrix is empty!" << endl;
        return Matrix();
    }
    Matrix<T> result(rows, cols, channel);
    if (!isSubMatrix && !b.isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            result.data[i] = data[i] + b.data[i];
        return result;
    }
    T *ptra = data;
    T *ptrb = b.data;
    T *ptr = result.data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            *ptr = *ptra + *ptrb;
            ++ptr;
            ++ptra;
            ++ptrb;
        }
        ptra = ptra + (step - cols) * channel;
        ptrb = ptrb + (b.step - b.cols) * b.channel;
    }
    return result;
}

template <class T>
Matrix<T> Matrix<T>::operator+(const T b) const
{
    if (data == NULL)
    {
        cerr << "The matrix is empty!" << endl;
        return Matrix();
    }
    Matrix result(rows, cols, channel);
    if (!isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            result.data[i] = data[i] + b;
        return result;
    }
    T *ptr = result.data;
    T *ptra = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            *ptr = *ptra + b;
            ++ptr;
            ++ptra;
        }
        ptra += (step - cols) * channel;
    }
    return result;
}

template <class T>
Matrix<T> Matrix<T>::operator-(const Matrix<T> &b) const
{
    if (rows != b.rows || cols != b.cols || channel != b.channel)
    {
        cerr << "Two matrixes aren't same size!" << endl;
        return Matrix();
    }
    if (data == NULL || b.data == NULL)
    {
        cerr << "The matrix is empty!" << endl;
        return Matrix();
    }
    Matrix<T> result(rows, cols, channel);
    if (!isSubMatrix && !b.isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            result.data[i] = data[i] - b.data[i];
        return result;
    }
    T *ptra = data;
    T *ptrb = b.data;
    T *ptr = result.data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            *ptr = *ptra - *ptrb;
            ++ptr;
            ++ptra;
            ++ptrb;
        }
        ptra += (step - cols) * channel;
        ptrb += (b.step - b.cols) * b.channel;
    }
    return result;
}

template <class T>
Matrix<T> Matrix<T>::operator-(const T b) const
{
    if (data == NULL)
    {
        cerr << "The matrix is empty!" << endl;
        return Matrix();
    }
    Matrix result(rows, cols, channel);
    if (!isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            result.data[i] = data[i] - b;
        return result;
    }
    T *ptr = result.data;
    T *ptra = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            *ptr = *ptra - b;
            ++ptr;
            ++ptra;
        }
        ptra += (step - cols) * channel;
    }
    return result;
}

template <class T>
Matrix<T> Matrix<T>::operator*(const Matrix<T> &b) const
{
    if (cols != b.rows || channel != b.channel)
    {
        cerr << "Can not multiple these two matrixes." << endl;
        return Matrix();
    }
    if (data == NULL || b.data == NULL)
    {
        cerr << "The matrix is empty!" << endl;
        return Matrix();
    }
    Matrix<T> result(rows, b.cols, channel);
    T *ptr;
    T *ptra;
    T *ptrb;
    for (int count = 0; count < channel; ++count)
    {
        ptr = result.data + count;
        ptra = data + count;
        ptrb = b.data + count;
        for (int i = 0; i < rows; ++i)
        {
            for (int k = 0; k < cols; ++k)
            {
                for (int j = 0; j < b.cols; ++j)
                {
                    *ptr += *ptrb * *ptra;
                    ptr += result.channel;
                    ptrb += b.channel;
                }
                ptr -= result.cols * result.channel;
                ptra += channel;
                ptrb += (b.step - b.cols) * b.channel;
            }
            ptra += (step - cols) * channel;
            ptrb = b.data + count;
            ptr += result.step * result.channel;
        }
    }
    return result;
}

template <class T>
Matrix<T> Matrix<T>::operator*(const T b) const
{
    if (data == NULL)
    {
        cerr << "The matrix is empty!" << endl;
        return Matrix();
    }
    Matrix result(rows, cols, channel);
    if (!isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            result.data[i] = data[i] * b;
        return result;
    }
    T *ptr = result.data;
    T *ptra = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols; ++j)
        {
            for (int k = 0; k < channel; ++k)
            {
                *ptr = *ptra * b;
                ++ptr;
                ++ptra;
            }
        }
        ptra += (step - cols) * channel;
    }
    return result;
}

template <class T>
bool Matrix<T>::operator==(const Matrix<T> &b) const
{
    if (rows != b.rows || cols != b.cols || channel != b.channel)
        return false;
    if (!isSubMatrix && !b.isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
        {
            if (data[i] != b.data[i])
                return false;
        }
        return true;
    }
    T *ptra = data;
    T *ptrb = b.data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            if (*ptra != *ptrb)
                return false;
            ++ptra;
            ++ptrb;
        }
        ptra += (step - cols) * channel;
        ptrb += (b.step - b.cols) * b.channel;
    }
    return true;
}

template <class T>
bool Matrix<T>::operator==(const T b) const
{
    if (!isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            if (data[i] != b)
                return false;
        return true;
    }
    T *ptr = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            if (*ptr != b)
                return false;
            ++ptr;
        }
        ptr += (step - cols) * channel;
    }
    return true;
}

template <class T>
bool Matrix<T>::operator!=(const Matrix<T> &b) const
{
    if (rows != b.rows || cols != b.cols || channel != b.channel)
        return true;
    if (!isSubMatrix && !b.isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
        {
            if (data[i] != b.data[i])
                return true;
        }
        return false;
    }
    T *ptra = data;
    T *ptrb = b.data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            if (*ptra != *ptrb)
                return true;
            ++ptra;
            ++ptrb;
        }
        ptra += (step - cols) * channel;
        ptrb += (b.step - b.cols) * b.channel;
    }
    return false;
}

template <class T>
bool Matrix<T>::operator!=(const T b) const
{
    if (!isSubMatrix)
    {
        for (int i = 0; i < rows * cols * channel; ++i)
            if (data[i] != b)
                return true;
        return false;
    }
    T *ptr = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            if (*ptr != b)
                return true;
            ++ptr;
        }
        ptr += (step - cols) * channel;
    }
    return false;
}

template <class T>
const Matrix<T> &Matrix<T>::operator=(const Matrix<T> &b)
{
    release();
    rows = b.rows;
    cols = b.cols;
    data = b.data;
    channel = b.channel;
    step = b.step;
    data_start = b.data_start;
    data_end = b.data_end;
    refcount = b.refcount;
    ++*refcount;
    isSubMatrix = b.isSubMatrix;
    return *this;
}

template <class T>
const Matrix<T> &Matrix<T>::operator=(const T b)
```

```cpp
{
    T *ptr = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            *ptr = b;
            ++ptr;
        }
        ptr += (step - cols) * channel;
    }
    return *this;
}

template <class T>
Matrix<T> Matrix<T>::subMatrix(int x, int y, int row, int col) const
{
    return Matrix(*this, x, y, row, col);
}

template <class T>
Matrix<T> &Matrix<T>::adjustROI(int dtop, int dbottom, int dleft, int dright)
{
    if (data == NULL)
        return *this;
    //locate the submatrix first
    int element = (data - data_start) / channel;
    int x = element / step + 1;
    int y = element + 1 - (x - 1) * step;
    int row = (data_end - data_start + 1) / channel / step;

    int leftBound, rightBound, upBound, lowBound;
    upBound = x - dtop;
    lowBound = x + rows - 1 + dbottom;
    leftBound = y - dleft;
    rightBound = y + cols - 1 + dright;
    if (lowBound < upBound || leftBound > rightBound)
    {
        cerr << "Can not adjust ROI!" << endl;
        return *this;
    }
    if (upBound <= 0)
        upBound = 1;
    if (lowBound > row)
        lowBound = row;
    if (leftBound <= 0)
        leftBound = 1;
    if (rightBound > step)
        rightBound = step;
    cols = rightBound - leftBound + 1;
    rows = lowBound - upBound + 1;
    data = data_start + ((upBound - 1) * step + leftBound - 1) * channel;
    if (rows == row && cols == step)
        isSubMatrix = false;
    return *this;
}

```

```cpp
template <class T>
Matrix<T> Matrix<T>::clone() const
{
    Matrix newMat(rows, cols, channel);
    if (!isSubMatrix)
    {
        memcpy(newMat.data, data, rows * cols * channel * sizeof(T));
        return newMat;
    }
    T *ptrdest = newMat.data;
    T *ptr = data;
    for (int i = 0; i < rows; ++i)
    {
        memcpy(ptrdest, ptr, cols * channel * sizeof(T));
        ptr += step * channel;
        ptrdest += newMat.step * channel;
    }
    return newMat;
}

template <class T>
void Matrix<T>::readFile(string file_path)
{
    ifstream ifs(file_path);
    if (!ifs.is_open())
    {
        cerr << "Can not open the file!" << endl;
        return;
    }
    T *ptr = data;
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols * channel; ++j)
        {
            if (!ifs.eof())
            {
                ifs >> *ptr;
                ++ptr;
            }
            else
            {
                ifs.close();
                return;
            }
        }
        ptr += (step - cols) * channel;
    }
    ifs.close();
    cout << "Read successfully!" << endl;
}

template <class T>
int Matrix<T>::getRows() const
{
    return rows;
}
template <class T>
int Matrix<T>::getCols() const
```

```cpp
{
    return cols;
}
template <class T>
int Matrix<T>::getChannel() const
{
    return channel;
}
template <class T>
bool Matrix<T>::getIsSubMatrix() const
{
    return isSubMatrix;
}
template <class T>
int Matrix<T>::getReferenceCount() const
{
    return *refcount;
}

template class Matrix<unsigned char>;
template class Matrix<short>;
template class Matrix<int>;
template class Matrix<float>;
template class Matrix<double>;
```

main.cpp

```cpp
#include "matrix.hpp"
#include <chrono>
#define TIME_START start = std::chrono::steady_clock::now();
#define TIME_END                \
    end = std::chrono::steady_clock::now();               \
    duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count(); \
    cout << "duration = " << duration << "ms" << endl;
int main()
{
    {
        //Test case1:
        cout << "Test case1:" << endl;
        Matrix<unsigned char> A(2, 4);
        Matrix<short> B(3, 2);
        Matrix<int> C(3, 4, 2);
        Matrix<float> D(2, 4, 3);
        Matrix<double> E(3, 5, 1);
        A.at(1, 3) = 'a';
        B.at(2, 1) = 3;
        C.at(2, 3, 2) = 100;
        D.at(2, 4, 2) = 3.14;
        E.at(2, 4, 1) = 5.56;
        cout << A << B << C << D << E;
    }

    //Test case2:
    {
        cout << "Test case2:" << endl;
        Matrix<int> A(3, 5);
        Matrix<int> B(A, 1, 2, 2, 2);
        Matrix<int> C(A, 1, 1, 3, 2);
    }

    //Test case3:
    {
        cout << "Test case3:" << endl;
        Matrix<int> B;
        Matrix<int> C;
        {

            Matrix<int> A(3, 5);
            B = A.subMatrix(1, 2, 2, 2);
            C = A.subMatrix(1, 1, 3, 2);
        }
    }

    Matrix<int> A(6, 7, 3);
    Matrix<int> B(5, 8, 3);
    for (int i = 1; i <= A.getRows(); ++i)
        for (int j = 1; j <= A.getCols(); ++j)
            for (int k = 1; k <= A.getChannel(); ++k)
                A.at(i, j, k) = 3 * i - j + 5 * k;
    for (int i = 1; i <= B.getRows(); ++i)
        for (int j = 1; j <= B.getCols(); ++j)
            for (int k = 1; k <= B.getChannel(); ++k)
                B.at(i, j, k) = 2 * i + j + 3 * k;

    //Test case4:
    {
        cout << "Test case4:" << endl;
        Matrix<int> C(A, 2, 4, 3, 2);
        Matrix<int> D = B.subMatrix(3, 5, 3, 3);
        cout << C << D;
    }

    //Test case5:
    {
        cout << "Test case5:" << endl;
        Matrix<int> C(A, 2, 5, 3, 4);
        cout << C;
    }

    //Test case6:
    {
        cout << "Test case6:" << endl;
        Matrix<int> C(A, 2, 2, 5, 5);
        cout << C;
        Matrix<int> D;
        D = C.subMatrix(3, 2, 2, 2);
        cout << D;
    }

    //Test case7:
```

```
    {
        cout << "Test case7:" << endl;
        Matrix<int> C(A, 3, 4, 2, 2);
        cout << C;
        C.adjustROI(1, 2, 1, 4);
        cout << C;
        C.adjustROI(-1, 3, 0, 0);
        cout << C;
    }

    //Test case8:
    {
        cout << "Test case8:" << endl;
        Matrix<int> C(A, 3, 4, 2, 2);
        cout << C;
        C.adjustROI(-2, -1, 0, 0);
        cout << C;
        C.adjustROI(0, 0, -1, -3);
        cout << C;
    }

    //Test case9:
    {
        cout << "Test case9:" << endl;
        Matrix<int> C(A, 2, 3, 3, 4);
        Matrix<int> D(B, 1, 1, 3, 4);
        cout << C << D;
        Matrix<int> E = C + D;
        cout << E;
        C.adjustROI(-1, 0, 0, 0);
        cout << C << D;
        cout << C + D;

        Matrix<int> F = 5 + C;
        cout << F;
    }

    //Test case10:
    {
        cout << "Test case10:" << endl;
        Matrix<int> C(A, 2, 3, 3, 4);
        Matrix<int> D(B, 1, 1, 3, 4);
        cout << C << D;
        Matrix<int> E = C - D;
        cout << E;
        C.adjustROI(-1, 0, 0, 0);
        cout << C << D;
        cout << C - D;

        Matrix<int> F = 30 - C;
        cout << F;
    }

    //Test case11:
    {
        cout << "Test case11:" << endl;
        Matrix<int> C(A, 2, 1, 3, 4);
        Matrix<int> D(B, 2, 3, 4, 5);
        cout << C << D;
        cout << C * D;
        D.adjustROI(0, -1, 0, 0);
        cout << C * D;
        cout << C * 2;
    }

    //Test case12:
    {
        cout << "Test case12:" << endl;
        Matrix<int> C(2, 3);
        C = 3;
        Matrix<int> D(2, 3);
        D = 3;
        cout << (C == D) << endl;
        cout << (C != D) << endl;
        cout << (A == B) << endl;
        cout << (A != B) << endl;
    }

    //Test case13:
    {
        cout << "Test case13:" << endl;
        Matrix<int> C(A, 2, 1, 3, 4);
        cout << C;
        Matrix<int> D = C.clone();
        cout << D;
        C.at(1, 1, 1) = 0;
        cout << C;
        cout << D;
    }

    // //Test speed:
    // auto start = std::chrono::steady_clock::now();
    // auto end = std::chrono::steady_clock::now();
    // auto duration = 0L;

    // Matrix<float> M(1024, 2048, 3);
    // Matrix<float> N(2048, 1024, 3);
    // M.readFile("1024-2048-3.txt");
    // N.readFile("2048-1024-3.txt");
    // TIME_START
    // Matrix<float> result = M * N;
    // TIME_END
    // ofstream ofs("out.txt");
    // ofs << result;
    // ofs.close();
    return 0;
}
```

> CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)

project(matrix)

aux_source_directory(. DIR_SRCS)

add_executable(matrix ${DIR_SRCS})
```

Part 5 - 总结

本次project重心主要放在了类的实现以及动态内存管理，对类与对象以及模板类有了更加深入的了解。同时，在内存管理上也遇到了些问题，也都在project的实现过程中逐一解决，也让我认识到内存管理的重要性（否则程序可能会中途崩溃）。此次project也最终实现了一个使用较为安全，也拥有基本功能的矩阵类，在未来仍可以进一步继续完善。

但由于时间限制，仍有一些想法还未来得及实现，包括使用多线程计算矩阵乘法、分别使用AVX和NEON指令集实现矩阵运算，并分别在arm平台上测试，日后有机会，仍然可以做出相应尝试，相信有不一样的认识！