CS205 C/C++ Programming - Project_4

Name: 刘乐奇（Liu Leqi）

SID： 12011327

1 题目分析

本程序需要用C++实现一个矩阵类，其中要包含矩阵的各种性质（如行数、列数、矩阵元素等等），以及对矩阵操作的一系列方法（如矩阵和，矩阵差，矩阵积，矩阵比较，矩阵拷贝等等）。

需要注意的是，该矩阵类需要适配矩阵元素的不同数据类型（如int，float等等），因此，在本程序中，将会用模板类来保证元素的数据类型的可变性，同时简化代码。

1.1 环境及工具

本程序主要使用Windows系统，利用vscode及wsl编写并编译运行，使用的C++语言标准为C++2a。同时也用了Clion。

此外本程序还在x86和arm中测试过，保证程序在不同环境的稳健性。

1.2 文件读取写入

在C++中，通过`<fstream>`文件流读取、输出文件内容是比较常用的。此外，若写入文件时目标路径下无指定文件，则创建之；否则将会报错，让用户检查是否误输入了同名文件。

1.3 矩阵合法性判断

由于矩阵内容和矩阵相乘需要符合一定条件，如内容需为数字、行列对应，因此在读取矩阵后需要对矩阵是否合法进行判断。若规范，则继续进行后续操作；否则直接退出程序。

1.4 计时器

可以通过`<util.hpp>`中的`Timer`类来计时，计算时间差并输出，能简便得出程序某部分的运行时间。

2 代码

本程序包含以下文件：

① util.hpp 文件检查，读取行列大小和矩阵数据，输出矩阵到文件中，复数类

② matrix.hpp 矩阵类

③ cal.hpp cal.cpp 矩阵计算

④ main.cpp, CMakeLists.txt

2.1 util

头文件

```
//util.hpp
#pragma once

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>

#include <iostream>
#include <iomanip>
#include <chrono>
#include <unistd.h>

#include "matrix.hpp"

#define TIMER Timer stopwatch;

void check_file(std::ifstream &);

size_t read_row(std::ifstream &);

size_t read_col(std::ifstream &);

template<typename T>
void from_file(matrix<T> &ma, const size_t row, const size_t col, std::ifstream
& f){
    T s;
    for (size_t i = 0; i < row; ++i) {
        std::string l;
        std::getline(f, l, '\n');
        std::stringstream ssl(l);
        for (size_t j = 0; j < col; ++j) {
            ssl >> s;
            ma.set(i, j, s);
        }
    }
    f.clear();
}

template<typename T>
void to_file(const matrix<T> &ma, std::ofstream &f){
    size_t row = ma.get_row();
    size_t col = ma.get_col();
    T tmp;
    for (size_t i = 0; i < row; ++i) {
        for (size_t j = 0; j < col; ++j) {
            f << std::setiosflags(std::ios::fixed) << std::setprecision(8) <<
ma.get_val()[i * col + j] << ' ';
        }
        f << '\n';
    }
    f.clear();
}

class Timer {
private:
    std::chrono::time_point<std::chrono::system_clock> start_;
    constexpr static auto unit_ =
            static_cast<double>(std::chrono::microseconds::period::num)
            / std::chrono::microseconds::period::den;
public:
    Timer() {
        this->start_ = std::chrono::system_clock::now();
    }

    ~Timer() {
        auto end = std::chrono::system_clock::now();
        std::cout << "Time spent: " << std::setiosflags(std::ios::fixed) <<
std::setprecision(8)
                  << static_cast<double>((end - this->start_).count()) * unit_
                  << " sec" << '\n';
    }
};


template<typename T>
class complex {
private:
    T re;
    T im;
public:
    complex() : re(0), im(0) {};

    complex(T re, T im) : re(re), im(im) {};

    complex<T> &operator+(const complex<T> &b) {
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        this->re += b.re;
        this->im += b.im;
        return *this;
    }

    complex<T> &operator-(const complex<T> &b) {
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        this->re -= b.re;
        this->im -= b.im;
        return *this;
    }

    complex<T> &operator*(const complex<T> &b) {
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        this->re = this->re * b.re - this->im * b.im;
        this->im = this->im * b.re + this->re * b.im;
        return *this;
    }

    complex<T> &operator/(const complex<T> &b) {
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        T base = sqrt(b.re * b.re + b.im * b.im);
        this->re = (this->re * b.re + this->im * b.im) / base;
        this->im = (this->im * b.re - this->re * b.im) / base;
        return *this;
    }

    bool operator==(const matrix<T> &b) {
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        return (this->re == b.re && this->im == b.im);
    }

    bool operator!=(const matrix<T> &b) {
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        return !this->operator==(b);
    }

    void show() {
        std::ios::sync_with_stdio(0), std::cin.tie(0), std::cout.tie(0);
        if (this == NULL) {
            std::cerr << "ERROR: Complex referred is NULL!" << '\n';
            throw std::invalid_argument("Complex referred is NULL!");
        }
        if (this->im == 0) std::cout << this->re << '\n';
        if (this->im < 0) std::cout << this->re << '-' << abs(this->im) << '\n';
        if (this->im > 0) std::cout << this->re << '+' << abs(this->im) << '\n';
    }
};
```

2.2 matrix

头文件

```
//matrix.hpp
#pragma once

#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdio>
#include <string>
#include <cmath>

template<typename T>
class matrix {
private:
    size_t row;
    size_t col;
    T *val;
    size_t refcount;
    size_t channel;

public:
    matrix() : row(0), col(0), val(NULL), refcount(1), channel(1) {};//default constructor
    matrix(size_t, size_t, size_t = 1);//initial with arguments
    matrix(const matrix<T> &);//copy
    ~matrix();

    size_t get_col() const;

    size_t get_row() const;

    T *get_val() const;

    T at(size_t, size_t);

    void set(size_t, size_t, T);

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

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

    friend matrix<T> &operator*(matrix<T> &ma, const T b) {
        if (ma.get_val() == NULL) {
            std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
            throw std::invalid_argument("Matrix referred is NULL!");
        }
        for (size_t i = 0; i < ma.get_row(); ++i)
            for (size_t j = 0; j < ma.get_col(); ++j)
                ma.get_val()[i * ma.get_col() + j] *= b;
        return ma;
    }

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

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

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

    void multiply(const matrix<T> &, const matrix<T> &, int = 1);

    void show();
};


template<typename T>
matrix<T>::matrix(size_t row, size_t col, size_t channel) {
    this->row = row;
    this->col = col;
    this->channel = channel;
    this->val = new T[row * col * channel];
    this->refcount = 1;
}

template<typename T>
matrix<T>::matrix(const matrix &ma) {
    this = ma;
    this->refcount++;
}

template<typename T>
matrix<T>::~matrix() {
    if (this == NULL || this->val == NULL) return;
    if (this->refcount > 1) this->refcount--;
    else delete[] this->val;
}

template<typename T>
size_t matrix<T>::get_row() const {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    return this->row;
}

template<typename T>
size_t matrix<T>::get_col() const {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    return this->col;
}
```

```
template<typename T>
T *matrix<T>::get_val() const {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    return this->val;
}

template<typename T>
T matrix<T>::at(size_t i, size_t j) {
    if (this == NULL || this->val == NULL) {
        std::cerr << "ERROR: Invalid matrix!" << '\n';
        throw std::invalid_argument("Invalid matrix!");
    }
    return this->val[i * this->col + j];
}

template<typename T>
void matrix<T>::set(size_t i, size_t j, T s) {
    this->val[i * this->col + j] = s;
}

template<typename T>
matrix<T> &matrix<T>::operator+(const matrix<T> &ma) {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    if (this->row != ma.get_row() || this->col != ma.get_col()) {
        std::cerr << "ERROR: Not match for addition!" << '\n';
        throw std::invalid_argument("Not match for addition!");
    }
    for (size_t i = 0; i < this->row; ++i)
        for (size_t j = 0; j < this->col; ++j)
            this->val[i * this->col + j] += ma.val[i * this->col + j];
    return *this;
}

template<typename T>
matrix<T> &matrix<T>::operator-(const matrix<T> &ma) {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    if (this->row != ma.get_row() || this->col != ma.get_col()) {
        std::cerr << "ERROR: Not match for subtraction!" << '\n';
        throw std::invalid_argument("Not match for subtraction!");
    }
    for (size_t i = 0; i < this->row; ++i)
        for (size_t j = 0; j < this->col; ++j)
            this->val[i * this->col + j] -= ma.val[i * this->col + j];
    return *this;
}

template<typename T>
matrix<T> &matrix<T>::operator=(const matrix<T> &ma) {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    if (this == &ma) return *this;
    this->row = ma.row;
    this->col = ma.col;
    this->val = ma.val;
    return *this;
}

template<typename T>
bool matrix<T>::operator==(const matrix<T> &ma) {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    if (this->row != ma.get_row() || this->col != ma.get_col())
        return false;
    for (size_t i = 0; i < this->row; ++i)
        for (size_t j = 0; j < this->col; ++j)
            if (this->val[i * this->col + j] != ma.val[i * this->col + j])
return false;
    return true;
}

template<typename T>
bool matrix<T>::operator!=(const matrix<T> &ma) {
    if (this == NULL) {
        std::cerr << "ERROR: Matrix referred is NULL!" << '\n';
        throw std::invalid_argument("Matrix referred is NULL!");
    }
    if (this->row != ma.get_row() || this->col != ma.get_col())
        return true;
    for (size_t i = 0; i < this->row; ++i)
        for (size_t j = 0; j < this->col; ++j)
            if (this->val[i * this->row + j] != ma.val[i * this->row + j])
return true;
    return false;
}

template<typename T>
void matrix<T>::multiply(const matrix<T> &ma, const matrix<T> &mb, int mode) {
    if (this == NULL) return;
    if (this->col != ma.get_row()) {
        std::cerr << "ERROR: Not match for multiplication!" << '\n';
        throw std::invalid_argument("Not match for multiplication!");
    }
    switch (mode) {
        case 1://simple
            simple(*this, ma, mb);
            break;
        case 2://simd
            simd(*this, ma, mb);
            break;
        case 3://blas
            blas(*this, ma, mb);
            break;
        default:
            std::cerr << "ERROR: invalid mode!" << '\n';
            throw std::invalid_argument("invalid mode!");
    }
}

template<typename T>
void matrix<T>::show() {
    std::ios::sync_with_stdio(0), std::cin.tie(0), std::cout.tie(0);
    for (size_t i = 0; i < this->row; ++i) {
        for (size_t j = 0; j < this->col; ++j) {
            std::cout << this->val[i * this->col + j];
            if (j != this->col) std::cout << ", ";
        }
        std::cout << ';' << '\n';
    }
}
```

源文件：无源文件，所有类及类函数都放在头文件中了。（原因具体见第**4**部分）

2.3 cal

头文件

```
//cal.hpp
#pragma once

#include "matrix.hpp"
#include <immintrin.h>
#include <cblas.h>

template<typename T>
void simple(matrix<T> &des, const matrix<T> &ma, const matrix<T> &mb);

float vec_dot(const float *, const float *, const size_t);

void simd(matrix<float> &des, const matrix<float> &ma, const matrix<float> &mb);

template<typename T>
void blas(matrix<T> &des, const matrix<T> &ma, const matrix<T> &mb);
```

2.4 main

```
//main.cpp
#include "matrix.hpp"
#include "cal.hpp"
#include "util.hpp"

int main(int argc, char **argv) {
    std::ios::sync_with_stdio(0), std::cin.tie(0), std::cout.tie(0);
    if (argc != 4) {
        std::cerr << "ERROR: The number of arguments is unexpected!" << '\n';
    }
    std::ifstream f1(argv[1]);
    std::ifstream f2(argv[2]);
    try {
        check_file(f1);
        check_file(f2);
    } catch (const std::invalid_argument &) {
        return -1;
    }
    size_t m1_row = read_row(f1);
    size_t m1_col = read_col(f1);
    size_t m2_row = read_row(f2);
    size_t m2_col = read_col(f2);

    matrix<float> ma(m1_row, m1_col);
    matrix<float> mb(m2_row, m2_col);
    from_file(ma, m1_row, m1_col, f1);
    from_file(mb, m2_row, m2_col, f2);

//    //测试拷贝函数、=重载、!=重载
//    matrix<float> mc(mb);
//    mc.show();
//    if (mc != ma) mc = ma;
//    mc.show();
//
//    mc.multiply(ma, mb, 1);
//    mc.multiply(ma, mb, 2);
//    mc.multiply(ma, mb, 3);
//    mc.show();

    return 0;
}
```

2.5 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16.3)
project(matmul)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_C_STANDARD 11)

set(CMAKE_CXX_FLAGS "-O3 -mavx -I /opt/OpenBLAS/include/ -L/opt/OpenBLAS/lib -lopenblas")

aux_source_directory(. DIR_SRC)

add_executable(matmul ${DIR_SRC})

target_link_libraries(matmul ${BLAS_LIBRARY})
```

有时候并未用上cmake，而是直接用命令行了（尤其在arm中）。

3 测试及分析

在测试中，x86是用本机测试，arm使用华为的云服务器测试。

1.1 文件读入输出

×86	32阶	256阶	2048阶
IN（两个矩阵）	2.87ms	91.79ms	1554.24ms
OUT（一个矩阵）	4.80ms	67.68ms	2331.66ms

arm	32阶	256阶	2048阶
IN（两个矩阵）	1.85ms	30.26ms	1431.11ms
OUT（一个矩阵）	1.69ms	19.88ms	1588.62ms

或许是本机还有其他进程在运行（比如网页，聊天工具，MobaXterm等等），在文件读取方面的速度非常不理想。

1.2 朴素矩阵乘法

	32阶	256阶	2048阶
×86	0.128	57.871	30210.730
arm	0.110	50.196	29061.677

没有特别的差别，速度大致差不多

1.3 OpenBlas

	32阶	256阶	2048阶
×86	0.778	30.007	2032.443
arm	0.310	20.196	1981.677

差异很明显，在arm下openBlas的增速非常多。

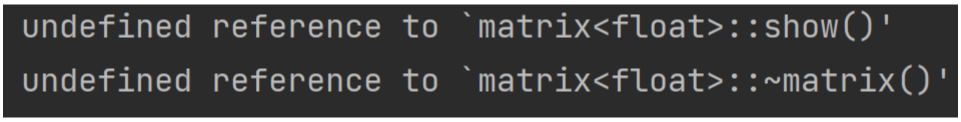
1.4 SIMD

没有测试成功，在×86和arm均未能运行，极有可能是代码出现问题。

4 困难及解决

4.1 模板类使用一些问题

在初次运行程序时，发生了如下这样的情况（已经确认该模板类方法已经定义了）：



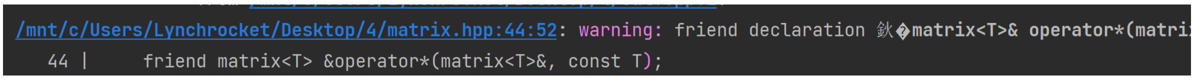
```
undefined reference to `matrix<float>::show()'
undefined reference to `matrix<float>::~matrix()'
```

在通过查阅网上的资料[1]得知，类模版并不是真正的类，它只是告诉编译器一种生成类的方法，编译器在遇到类模版的实例化时，就会按照模版生成相应的类。而每一个cpp文件是独立编译的，那么如果将类模版的成员函数单独放在一个cpp文件中，编译器便无法确定要根据什么类型来产生相应的类，也就造成了错误。

网上给出的方法是，一般把类函数的定义也下写入.h或.hpp文件中。

4.1 模板类中的友元函数

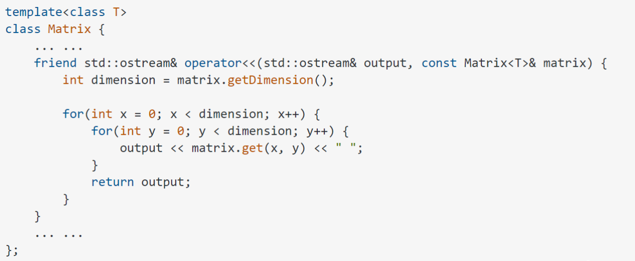
在调用该函数并编译时，编译器说我未定义该函数。



```
/mnt/c/Users/Lynchrocket/Desktop/4/matrix.hpp:44:52: warning: friend declaration 鈥�matrix<T>& operator*(matri
   44 |     friend matrix<T> &operator*(matrix<T>&, const T);
```

在通过查阅网上的资料[2]得知，该问题有两种解决方法。一种是直接将该函数的定义与声明均移至类中；另一种则比较麻烦，具体如下：

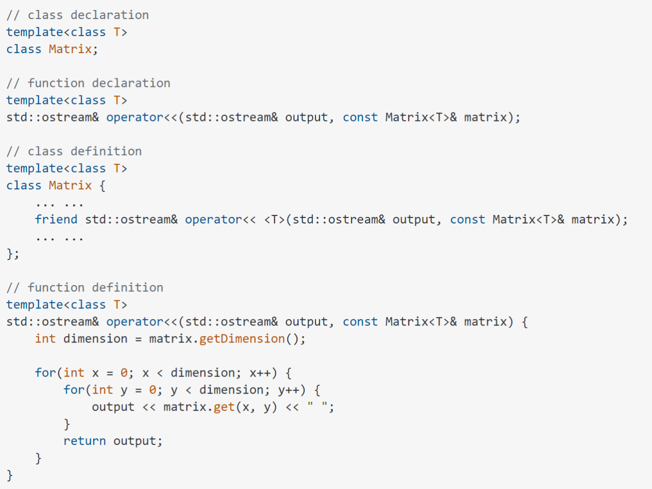
法1：



```cpp
template<class T>
class Matrix {
    ... ...
    friend std::ostream& operator<<(std::ostream& output, const Matrix<T>& matrix) {
        int dimension = matrix.getDimension();

        for(int x = 0; x < dimension; x++) {
            for(int y = 0; y < dimension; y++) {
                output << matrix.get(x, y) << " ";
            }
            return output;
        }
    }
    ... ...
};
```

法2：



```cpp
// class declaration
template<class T>
class Matrix;

// function declaration
template<class T>
std::ostream& operator<<(std::ostream& output, const Matrix<T>& matrix);

// class definition
template<class T>
class Matrix {
    ... ...
    friend std::ostream& operator<< <T>(std::ostream& output, const Matrix<T>& matrix);
    ... ...
};

// function definition
template<class T>
std::ostream& operator<<(std::ostream& output, const Matrix<T>& matrix) {
    int dimension = matrix.getDimension();

    for(int x = 0; x < dimension; x++) {
        for(int y = 0; y < dimension; y++) {
            output << matrix.get(x, y) << " ";
        }
        return output;
    }
}
```

在本程序中，选用了第一种方法。

6 总结

即便是同一个程序，在不同的系统中也有不同的表现。通过这次对x86和arm平台上矩阵乘法程序的对比，对这个道理越发理解。

[1] 模板使用的问题：https://blog.csdn.net/breakpoints_/article/details/80565452 ↩

[2] 模板类中的友元：https://stackoverflow.com/questions/48626437/friend-and-template-in-c ↩