

**课 程 实 验 报 告**

**课程名称：C++程序设计**

**实验名称：面向对象的矩阵运算编程**

**院 系 ：计算机科学与技术**

**专业班级 ： CS2011**

**学 号 ： U202010755**

**姓 名 ： 路昊东**

**指导教师 ： 纪俊文**

**2021 年 11 月 23 日**

1. **需求分析**
2. **题目要求**

**矩阵MAT是行列定长的二维数组。常见的矩阵运算包括矩阵的加、减、乘、转置和赋值等运算。请对矩阵MAT类中的所有函数成员编程，并对随后给出的main( )函数进行扩展，以便完成矩阵及其重载的所有运算符的测试。输出矩阵元素时整数用”%6ld” 或”%6lld”打印，浮点数用”%8f”或”%8lf”打印。至少要测试两种实例类MAT<int>和MAT<long long>。**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iomanip>**

**#include <exception>**

**#include <typeinfo>**

**#include <string.h>**

**using namespace std;**

**template <typename T>**

**class MAT {**

**T\* const e; //指向所有整型矩阵元素的指针**

**const int r, c; //矩阵的行r和列c大小**

**public:**

**MAT(int r, int c); //矩阵定义**

**MAT(const MAT& a); //深拷贝构造**

**MAT(MAT&& a)noexcept; //移动构造**

**virtual ~MAT()noexcept;**

**virtual T\* const operator[ ](int r);//取矩阵r行的第一个元素地址，r越界抛异常**

**virtual MAT operator+(const MAT& a)const; //矩阵加法，不能加抛异常**

**virtual MAT operator-(const MAT& a)const; //矩阵减法，不能减抛异常**

**virtual MAT operator\*(const MAT& a)const; //矩阵乘法，不能乘抛异常**

**virtual MAT operator~()const; //矩阵转置**

**virtual MAT& operator=(const MAT& a); //深拷贝赋值运算**

**virtual MAT& operator=(MAT&& a)noexcept; //移动赋值运算**

**virtual MAT& operator+=(const MAT& a); //“+=”运算**

**virtual MAT& operator-=(const MAT& a); //“-=”运算**

**virtual MAT& operator\*=(const MAT& a); //“\*=”运算**

**//print输出至s并返回s：列用空格隔开，行用回车结束**

**virtual char\* print(char\* s)const noexcept;**

**};**

**int main(int argc, char\* argv[ ]) //请扩展main()测试其他运算**

**{**

**MAT<int> a(1, 2), b(2, 2), c(1, 2);**

**char t[2048];**

**a[0][0] = 1; //类似地初始化矩阵的所有元素**

**a[0][1] = 2; //等价于“\*(a.operator[ ](0)+1)=2;”即等价于“\*(a[0]+1)=2;”**

**a.print(t); //初始化矩阵后输出该矩阵**

**b[0][0] = 3; b[0][1] = 4; //调用T\* const operator[ ](int r)初始化数组元素**

**b[1][0] = 5; b[1][1] = 6;**

**b.print(t);**

**c = a \* b; //测试矩阵乘法运算**

**c.print(t);**

**(a + c).print(t); //测试矩阵加法运算**

**c = c - a; //测试矩阵减法运算**

**c.print(t);**

**c += a; //测试矩阵“+=”运算**

**c.print(t);**

**c = ~a; //测试矩阵转置运算**

**c.print(t);**

**return 0;**

**}**

1. **需求分析**

本实验要求实现一个整型的矩阵及对其进行操作的函数的类MAT，类中成员包括行和列，以及指向矩阵成员的指针。包括矩阵的初始化、深拷贝初始化、移动初始化，运算操作、深拷贝赋值、移动赋值以及打印删除等功能，矩阵的内容存储在一个二维数组中。要求写成模板的形式，至少实现实例类<int>和<long long>，细节分析如下：

（1）**MAT(int r, int c)**对矩阵进行初始化，若r与c均大于零，表示行列数合法，为成员e分配r\*c个T类型元素的空间内存并对成员r与c赋初值，否则令成员e为nullptr，成员r与c为0。

（2）**MAT(const MAT& a)**对矩阵进行深拷贝构造，在用已经存在的对象a深拷贝构造新对象时，新对象的成员e不能共用a.e的内存，而应为新对象的成员e分配与对象a的成员e同样大小的内存，并将对象a成员e相应的内容深拷贝至新对象为对应e分配的内存；同时，新对象成员r与成员e应该设置为与对象a的成员r与成员e相同。

（3）**MAT(MAT&& a)noexcept**对矩阵进行移动拷贝构造，在用已经存在的对象a移动构造新对象时，新对象接受使用a为其成员e分配的内存，并且新对象的r与c应设置为与对象a的成员r与成员e相同；最后，应将对象a的成员e设置为空指针，同时将其对应的r与c设置为0。

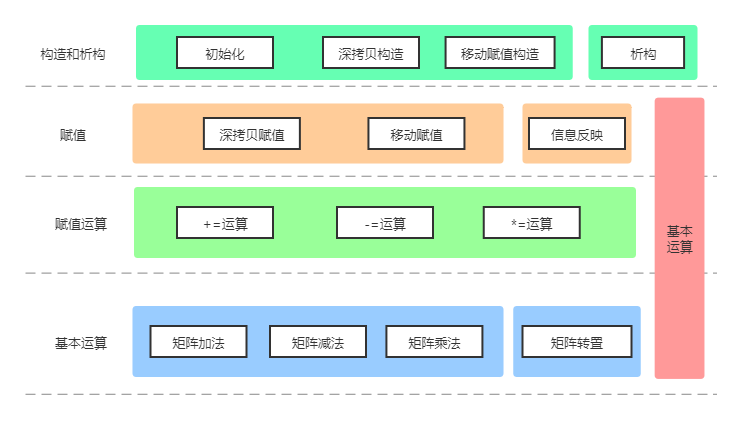
（4）**virtual ~MAT()noexcept**为析构函数，析构时，若指针成员e不为空指针，则应将e所指向的空间内存释放，同时，将成员r与成员c设置为0。

（5）**virtual T\* const operator[](int r)**功能为取矩阵r行的第一个元素地址，首先应判断r是否越界，若r越界，则应抛出异常，否则，返回矩阵r行的第一个元素地址。

（6）**virtual MAT operator+(const MAT& a)const**实现矩阵加法，若两个矩阵行列数不同，则不能加，应抛出异常，否则，按照矩阵加法运算方法进行两个矩阵的加法运算，返回相加之后的矩阵。

（7）**virtual MAT operator-(const MAT& a)const**实现矩阵减法，若两个矩阵行列数不同，则不能减，应抛出异常，否则，按照矩阵减法运算方法进行两个矩阵的减法运算，返回相减之后的矩阵。

（8）**virtual MAT& operator=(const MAT& a)**为深拷贝赋值函数，在用等号右边的对象a深拷贝赋值等号左边的对象时，等号左边的对象的成员e不能共用对象a为成员e分配的内存，若等号左边的对象为成员e分配了内存，则应先释放掉以避免内存泄漏，然后为e分配和对象a的成员e相同大小的内存，并且将对象a的成员e的内容拷贝至新对象的成员e的内存。同时，等号左边对象的成员r与c应设置为和a中成员r和c的对应值相同。

（9）**virtual MAT& operator=(MAT&& a)noexcept**为移动赋值运算，在用等号右边的对象a移动赋值给等号左边的对象时，等号左边的对象如果已经为其成员e分配了内存，则应先释放以避免内存泄漏，然后接受使用a为e分配的内存，并且等号左边对象中的r与c应和a中的成员r与c的对应 值相同；对象a的成员e应设置为空指针表示内存被移走，同时其对应的r与c应设置为0。

1. **系统设计**
2. **概要设计**

类中成员包括行和列，以及指向矩阵成员的指针。用长度为r\*c的一维数组来模拟行数为r、列数为c的二维数组。本系统主要分为以下几个模块：矩阵的创建与赋值、常见的矩阵运算包括矩阵的加、减、乘、转置和赋值等运算。

1. **详细设计**

由于MAT是类模板，需要用到泛型“template<class T>”。

1. **MAT(int r, int c);**

功能：对矩阵进行初始化，根据给定的r和c构造大小为[r][c]的矩阵。

实现：若r与c均大于零，表示行列数合法，为成员e分配r\*c个T类型元素的空间内存并对成员r与c赋初值，否则令成员e为空指针，成员r与c为0，矩阵初始化完成。

1. **MAT(const MAT& a);**

功能：根据已有的矩阵a，对矩阵进行深拷贝构造。

实现：要求用已经存在的对象a深拷贝构造新对象时，新对象的成员e不能共用a.e的内存，而应为新对象的成员e分配与对象a的成员e同样大小的内存，并将对象a成员e相应的内容深拷贝至新对象为对应e分配的内存；同时，新对象成员r与成员e应该设置为与对象a的成员r与成员e相同。因此先给成员e分配大小为a.r\*a.c的内存空间，遍历a中所有元素赋给新对象的对应下标的元素，然后将r与c分别设置为a.r与a.c。

1. **MAT(MAT&& a)noexcept;**

功能：对矩阵进行移动拷贝构造。

实现：要求用已经存在的对象a移动构造新对象时，新对象接受使用a为其成员e分配的内存，并且新对象的r与c应设置为与对象a的成员r与成员e相同；最后，应将对象a的成员e设置为空指针，同时将其对应的r与c设置为0。因此直接将e指向a.e，然后将r与c分别设置为a.r与a.c,将a.e设置为空指针，最后将a.r与a.c设置为0.由于e,r,c都是只读变量，因此修改其值时需要用const\_cast进行强制类型转换。

1. **virtual ~MAT()noexcept;**

功能：析构函数。

实现：析构时，若指针成员e不为空指针，则应将e所指向的空间内存释放，同时，将成员r与成员c设置为0。

1. **virtual T\* const operator[](int r);**

功能：取矩阵r行的第一个元素地址。

实现：首先应判断r是否越界，若r越界，则应抛出异常，否则，返回矩阵r行的第一个元素地址。

1. **virtual MAT operator+(const MAT& a)const;**

功能：矩阵加法。

实现：若两个矩阵行列数不同，则不能加，应抛出异常，否则，按照矩阵加法运算方法进行两个矩阵的加法运算，返回相加之后的矩阵。

1. **virtual MAT operator-(const MAT& a)const;**

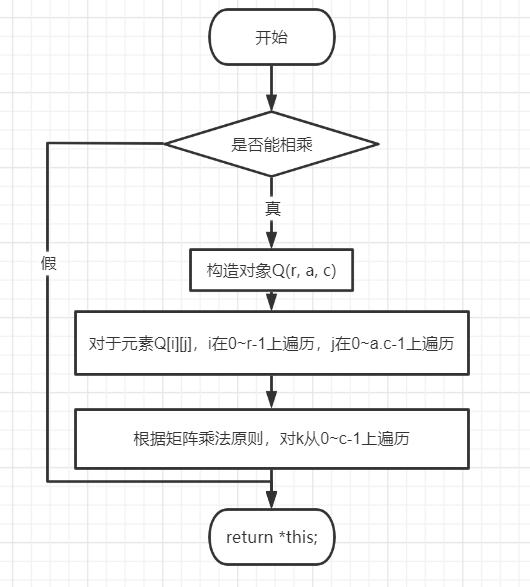
功能：矩阵减法。

实现：若两个矩阵行列数不同，则不能减，应抛出异常，否则，按照矩阵减法运算方法进行两个矩阵的减法运算，返回相减之后的矩阵。

1. **virtual MAT operator\*(const MAT& a)const;**

功能：矩阵乘法。

实现：首先需要判断两个矩阵能不能相乘，即判断c是否等于a.r。若c不等于a.r，则两个矩阵不满足相乘条件，不能相乘，抛出异常。否则可以相乘，进行矩阵乘法运算，最后返回相乘后的矩阵，函数流程图如下：



1. **virtual MAT operator~()const;**

功能：矩阵转置。

实现：可以构造一个行列数为c、r的矩阵，且第i行j列对应的值，与原矩阵j行i列对应的值相等，最后返回该矩阵。

1. **virtual MAT& operator=(const MAT& a);**

功能：深拷贝赋值函数。

实现：在用等号右边的对象a深拷贝赋值等号左边的对象时，等号左边的对象的成员e不能共用对象a为成员e分配的内存，若等号左边的对象为成员e分配了内存，则应先释放掉以避免内存泄漏，然后为e分配和对象a的成员e相同大小的内存，并且将对象a的成员e的内容拷贝至新对象的成员e的内存。同时，等号左边对象的成员r与c应设置为和a中成员r和c的对应值相同。由于成员e,r,c都是只读变量，因此修改其值时需要用const\_cast进行强制类型转换。

1. **virtual MAT& operator=(MAT&& a)noexcept;**

功能：移动赋值运算。

实现：在用等号右边的对象a移动赋值给等号左边的对象时，等号左边的对象如果已经为其成员e分配了内存，则应先释放以避免内存泄漏，然后接受使用a为e分配的内存，并且等号左边对象中的r与c应和a中的成员r与c的对应 值相同；对象a的成员e应设置为空指针表示内存被移走，同时其对应的r与c应设置为0。由于成员e,r,c都是只读变量，因此修改其值时需要用const\_cast进行强制类型转换。

1. **virtual MAT& operator+=(const MAT& a);**

功能：矩阵加法赋值运算。

实现：仿照之前思路或者直接调用之前定义过的深拷贝赋值与矩阵加法运算，直接返回\*this = \*this + a。

1. **virtual MAT& operator-=(const MAT& a);**

功能：矩阵减法赋值运算。

实现：仿照之前思路或者直接调用之前定义过的深拷贝赋值与矩阵减法运算，直接返回\*this = \*this - a。

1. **virtual MAT& operator\*=(const MAT& a);**

功能：矩阵乘法赋值运算。

实现：仿照之前思路或者直接调用之前定义过的深拷贝赋值与矩阵乘法运算，直接返回\*this = \*this \* a。

1. **virtual char\* print(char\* s)const noexcept;**

功能：输出矩阵至s并返回s。

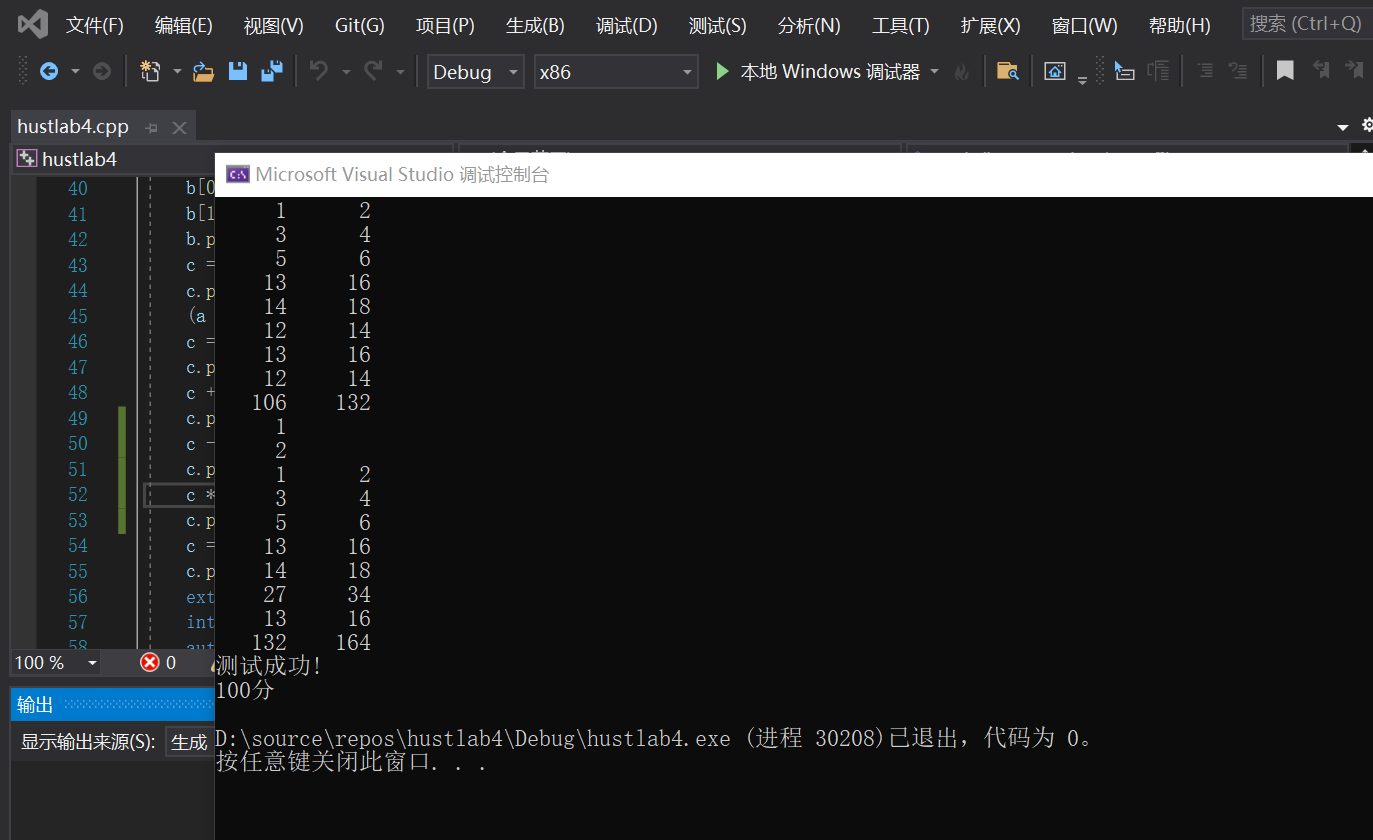
实现：首先考察元素的typeid。然后再根据typeid的不同，进行相应的存储。列用空格隔开，行用回车结束。可以用到sprintf\_s函数将整型值转换为字符串，存入s中，在每一行结束之前，每个整型值之后加上空格，当一行结束之时，存入换行符。

1. **软件开发**

使用Visual Studio 2019进行开发与调试。编译模式为X86。

1. **软件测试.**

在属性中添加库目录，在链接器中添加附加依赖项TestMAT.lib，使用测试库函数const char\* TestMAT(int& s)自动测试函数进行测试，测试结果为100分。



1. **特点与不足**
2. **技术特点**
3. 使用类模板，运用泛型知识；
4. 使用一维数组模拟实现二维数组。
5. **不足和改进的建议**

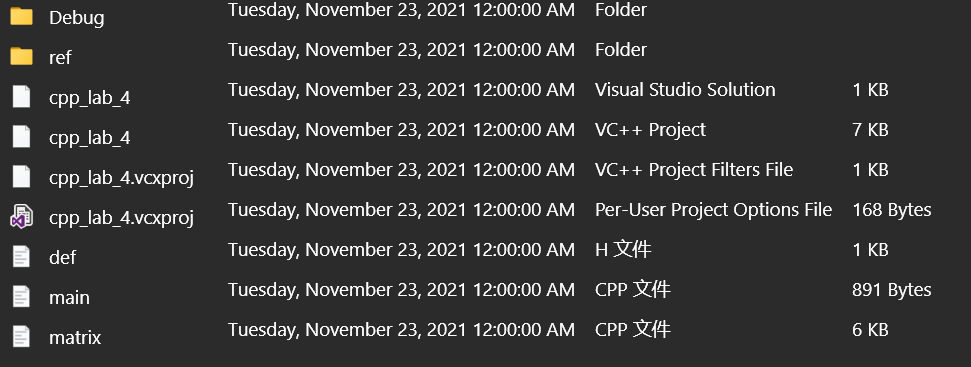
实验需求中涉及到矩阵的一些具体运算的功能方法比较少，且涉及到的优化比较少。

1. **过程和体会**
2. **遇到的主要问题和解决方法**
3. 编写矩阵乘法函数时开始由于对于矩阵乘法有些忘记，搞错了行和列。
4. 注意到深拷贝赋值和构造时，由于函数memcpy与memmove都是浅拷贝，对于类模板，若T类型为类或指针，其往往无法做到深拷贝其内存，故采用for循环进行深拷贝。
5. **课程设计的体会**

本次实验后，我对于使用泛型模板进行类的深拷贝构造、移动构造、深拷贝赋值运算与移动赋值运算等操作有了更深的了解，同时也对于矩阵的基本运算加深印象，课后自主拓展学习泛型的有关知识，收益匪浅。同时使用google开源代码规范书写，学习提高代码美观度。

1. **源码和说明**
2. **文件清单及其功能说明**

def.h是矩阵的定义，matrix.cpp是矩阵的功能实现，main.cpp是测试程序。



1. **用户使用说明书**

使用VS2019 链接测试库后，按Ctrl + F5可直接运行。

入门使用技巧：

1.运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单

2.调试程序: F5 或调试 >“开始调试”菜单

3.使用解决方案资源管理器窗口添加/管理文件

4.使用团队资源管理器窗口连接到源代码管理

5.使用输出窗口查看生成输出和其他消息

6.使用错误列表窗口查看错误

7.转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项” 以将现有代码文件添加到项目

8.将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件

1. **源代码**

**--------------------------------------------------------def.h----------------------------------------------------------**

#pragma once

#ifndef \_\_DEF\_H\_\_

#define \_\_DEF\_H\_\_

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iomanip>

#include <exception>

#include <typeinfo>

#include <string.h>

#include <iostream>

using namespace std;

template <typename T>

class MAT {

    T\* const e;                                 // 指向所有整型矩阵元素的指针

    const int r, c;                             // 矩阵的行r和列c大小

 public:

    MAT(int r, int c);                          // 矩阵定义

    MAT(const MAT& a);                          // 深拷贝构造

    MAT(MAT&& a)noexcept;                       // 移动构造

    virtual ~MAT()noexcept;

    virtual T\* const operator[ ](int r);        // 取矩阵r行的第一个元素地址，r越界抛异常

    virtual MAT operator+(const MAT& a)const;   // 矩阵加法，不能加抛异常

    virtual MAT operator-(const MAT& a)const;   // 矩阵减法，不能减抛异常

    virtual MAT operator\*(const MAT& a)const;   // 矩阵乘法，不能乘抛异常

    virtual MAT operator~()const;               // 矩阵转置

    virtual MAT& operator=(const MAT& a);       // 深拷贝赋值运算

    virtual MAT& operator=(MAT&& a)noexcept;    // 移动赋值运算

    virtual MAT& operator+=(const MAT& a);      // “+=”运算

    virtual MAT& operator-=(const MAT& a);      // “-=”运算

    virtual MAT& operator\*=(const MAT& a);      // “\*=”运算

    virtual char\* print(char\* s)const noexcept; // print输出至s并返回s：列用空格隔开，行用回车结束

};

template MAT<int>;

template MAT<long long>;

#endif // !\_\_DEF\_H

**--------------------------------------------------------matrix.cpp--------------------------------------------------**

#include"def.h"

/// <summary>

/// 矩阵定义

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="r"></param>

/// <param name="c"></param>

template<typename T>

MAT<T>::MAT(int r, int c) : r(r), c(c), e(new T[r \* c]) {}

/// <summary>

/// 深拷贝构造

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

template<typename T>

MAT<T>::MAT(const MAT& a) : e(new T[a.r \* a.c]), r(a.r), c(a.c) {

    for (int i = 0; i < a.r \* a.c; ++i) {

        e[i] = a.e[i];

    }

}

/// <summary>

/// 移动构造

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T>::MAT(MAT&& a) noexcept : e(a.e), r(a.r), c(a.c) {

    \*(T\*)&(a.c) = 0;

    \*(T\*)&(a.r) = 0;

    \*(T\*\*)&(a.e) = nullptr;

}

/// <summary>

/// 销毁矩阵

/// </summary>

/// <typeparam name="T"></typeparam>

/// <returns></returns>

template<typename T>

MAT<T>::~MAT() noexcept {

    if (e != nullptr) {

        delete[] e;

    }

    \*(T\*\*)&e = nullptr; // 将剩余变量赋值为0

    \*(int\*)&r = 0;

    \*(int\*)&c = 0;

}

/// <summary>

/// 取矩阵r行的第一个元素地址，r越界抛异常

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="r"></param>

/// <returns></returns>

template<typename T>

T\* const MAT<T>::operator[] (int r) {

    if (r < this->r && r >= 0) {

        return &e[r \* c];

    }else {

        throw("Operator[] ERROR!");

    }

}

/// <summary>

/// 矩阵加法，不能加抛异常

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T> MAT<T>::operator+(const MAT& a) const {

    if (r == a.r && c == a.c) {

        MAT R(r, c);

        int tmp = 0;

        int i = 0;

        for (i = 0; i < r \* c; i++){

            R.e[i] = e[i] + a.e[i];

        }

        return R;

    }

    throw("Operator+ ERROR!");       // 抛出异常

}

/// <summary>

/// 矩阵减法，不能减抛异常

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T> MAT<T>::operator-(const MAT& a) const {

    if (r == a.r && c == a.c) {

        MAT R(r, c);

        int i = 0;

        for (i = 0; i < r \* c; i++) {

            R.e[i] = e[i] - a.e[i]; // 相减

        }

        return R;

    }

    throw("Operator- ERROR!");      // 抛出异常

}

/// <summary>

/// 矩阵乘法，不能乘抛异常

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T> MAT<T>::operator\*(const MAT& a) const {

    if (c == a.r) { // 当this的列数等于a的行数时

        MAT ans(r, a.c);

        for (int i = 0; i < r; ++i) {

            for (int j = 0; j < a.c; ++j) {

                ans[i][j] = 0;

                for (int k = 0; k < c; ++k) {

                    ans[i][j] = ans[i][j] + e[i \* c + k] \* a.e[k \* c + j];

                }

            }

        }

        return ans;

    }

    throw("Operator\* ERROR!");

}

/// <summary>

/// 矩阵转置

/// </summary>

/// <typeparam name="T"></typeparam>

/// <returns></returns>

template<typename T>

MAT<T> MAT<T>::operator~() const {

    MAT ans(c, r);

    for (int i = 0; i < r; ++i) {

        for (int j = 0; j < c; ++j) {

            ans[j][i] = e[i \* c + j];

        }

    }

    return ans;

}

/// <summary>

/// 深拷贝赋值运算

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T>& MAT<T>::operator=(const MAT& a) {

    if (this != &a) {       // 若this和a不相等则进行操作

        if (e != nullptr) { // 如果p中队列不为空

            delete[] e;

        }

        \*(T\*\*)&e = new T[r \* c];

        for (int i = 0; i < r \* c; ++i) {

            e[i] = a.e[i];

        }

        int\* tmp;

        tmp = (int\*)&r;

        \*tmp = a.r;

        tmp = (int\*)&c;

        \*tmp = a.c;

    }

    return \*this;

}

/// <summary>

/// 移动赋值运算

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T>& MAT<T>::operator=(MAT<T>&& a)noexcept {

    if (this != &a) {       // 若p和q不相等则进行操作

        if (e != nullptr){  // 如果p中队列不为空

            delete[] e;     // 则释放内存，避免泄露

        }

        \*(T\*\*)&e = a.e;     // 进行相应浅拷贝赋值

        \*(int\*)&r = a.r;

        \*(int\*)&c = a.c;

        \*(T\*\*)&a.e = nullptr;// 将原来结构体赋值为0

        \*(int\*)&a.r = 0;

        \*(int\*)&a.c = 0;

        return \*this;

    }

}

/// <summary>

/// “+=”运算

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T>& MAT<T>::operator+=(const MAT<T>& a) {

    if (r == a.r && c == a.c) {

        int i = 0;

        for (i = 0; i < r \* c; ++i) {

            e[i] += a.e[i];

        }

        if (i == r \* c) {       // 检查循环是否完整结束

            return \*this;

        }

    }

    throw("Operator+= ERROR!"); // 抛出异常

}

/// <summary>

/// “-=”运算

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T>& MAT<T>::operator-=(const MAT<T>& a) {

    if (r == a.r && c == a.c) {

        int i = 0;

        for (i = 0; i < r \* c; i++) {

            e[i] -= a.e[i];

        }

        if (i == r \* c) {       // 检查循环是否完整结束

            return \*this;

        }

    }

    throw("Operator-= ERROR!"); // 抛出异常

 }

/// <summary>

/// “\*=”运算

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="a"></param>

/// <returns></returns>

template<typename T>

MAT<T>& MAT<T>::operator\*=(const MAT<T>& a) {

    if (c == a.r) {

        MAT ans(r, a.c);

        for (int i = 0; i < r; ++i) {

            for (int j = 0; j < a.c; ++j) {

                ans[i][j] = 0;

                for (int k = 0; k < c; ++k) {

                    ans[i][j] = ans[i][j] + e[i \* c + k] \* a.e[k \* c + j];

                }

            }

        }

        (\*this) = ans;

        return \*this;

    }

    throw("Operator\*= ERROR!");

}

/// <summary>

/// print输出至s并返回s：列用空格隔开，行用回车结束

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="s"></param>

/// <returns></returns>

template<typename T>

char\* MAT<T>::print(char\* s)const noexcept {

    int n = 0;

    if (typeid(e[0]) == typeid(int)) {

        for (int i = 0; i < r; i++) {

            for (int j = 0; j < c; j++) {

                if (j != 0) {

                    n += sprintf(s + n, "%c", ' ');

                }

                n += sprintf(s + n, "%6ld", e[i \* c + j]);

                if (j == c - 1) {

                    n += sprintf(s + n, "%c", '\n');

                }

            }

        }

    } else if (typeid(e[0]) == typeid(long)) {

        for (int i = 0; i < r; i++) {

            for (int j = 0; j < c; j++) {

                if (j != 0) {

                    n += sprintf(s + n, "%c", ' ');

                }

                n += sprintf(s + n, "%6lld", e[i \* c + j]);

                if (j == c - 1) {

                    n += sprintf(s + n, "%c", '\n');

                }

            }

        }

    } else if (typeid(e[0]) == typeid(float)) {

        for (int i = 0; i < r; i++) {

            for (int j = 0; j < c; j++) {

                if (j != 0) {

                    n += sprintf(s + n, "%c", ' ');

                }

                n += sprintf(s + n, "%8f", e[i \* c + j]);

                if (j == c - 1) {

                    n += sprintf(s + n, "%c", '\n');

                }

            }

        }

    } else if (typeid(e[0]) == typeid(double)) {

        for (int i = 0; i < r; i++) {

            for (int j = 0; j < c; j++) {

                if (j != 0) {

                    n += sprintf(s + n, "%c", ' ');

                }

                n += sprintf(s + n, "%8lf", e[i \* c + j]);

                if (j == c - 1) {

                    n += sprintf(s + n, "%c", '\n');

                }

            }

        }

    }

    s[n] = '\0';

    printf("%s", s);

    return s;

 }

**--------------------------------------------------------main.cpp---------------------------------------------------**

#include "def.h"

int main(int argc, char\* argv[]) {

MAT<int> a(1, 2), b(2, 2), c(1, 2);

char t[2048];

a[0][0] = 1; // 类似地初始化矩阵的所有元素

a[0][1] = 2; // 等价于“\*(a.operator[ ](0)+1)=2;”即等价于“\*(a[0]+1)=2;”

a.print(t); // 初始化矩阵后输出该矩阵

b[0][0] = 3; b[0][1] = 4; // 调用T\* const operator[ ](int r)初始化数组元素

b[1][0] = 5; b[1][1] = 6;

b.print(t);

c = a \* b; // 测试矩阵乘法运算

c.print(t);

(a + c).print(t); // 测试矩阵加法运算

c = c - a; // 测试矩阵减法运算

c.print(t);

c += a; // 测试矩阵“+=”运算

c.print(t);

c = ~a; // 测试矩阵转置运算

c.print(t);

extern const char\* TestMAT(int& s); // 引入测试函数

int nMark = 0; // 记录分数

const char\* strResult = TestMAT(nMark); // 记录提示语句

cout << strResult << endl;

cout << nMark << endl;

return 0;

}