

华中科技大学

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：计卓 2101

学 号：U202112071

姓 名：王彬

指导教师：金良海

2022 年 12 月 13 日

一、需求分析

1. 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，最后一行用换行符结束 "\n"。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;
    }

```

2. 需求分析

采用面向对象的方式实现矩阵的基本操作。这些操作包括，无参数地构造矩阵，用一个矩阵初始化另一个矩阵，对矩阵进行加减乘与转置操作，输出矩阵中元素，深拷贝赋值矩阵，销毁矩阵等操作。

二、系统设计

1. 概要设计

我们定义模板矩阵类为 $\text{MAT}<\text{T}>$ ，它具有一个指向类型为 T 的 `const` 指针 e ，和分别描述矩阵的行数和列数的 `const` 整型常量 r 和 c 。

在这些私有数据成员之外，我们为模板类定义一系列构造矩阵，用一个矩阵初始化另一个矩阵，对矩阵进行加减乘与转置操作，输出矩阵中元素，深拷贝赋值矩阵，销毁矩阵等操作。这样可以实现一个矩阵的基本操作。

2. 详细设计

(1) 参数为 (int, int) 的矩阵定义构造函数：

传入矩阵的行数和列数 r 及 c ，初始化矩阵。将 e 指向对应大小的矩阵空间，并相应地初始化矩阵的行数列数。同时，将矩阵中的每一个元素初始化为零。

(2) 深拷贝构造函数 $\text{MAT}<\text{T}>::\text{MAT}(\text{const MAT}\& \text{a})$:

传入一个矩阵的引用，并以此对本矩阵进行构造。将该矩阵的长宽和每一个元素的数量大小都赋值为原矩阵的数值。并为该矩阵开辟相应的空间。

(3) 移动构造函数 $\text{MAT}<\text{T}>::\text{MAT}(\text{MAT}\&\& \text{a})$:

传入一个右值引用，并将原右值引用的指针赋给该矩阵中的对应元素后，将右值引用对象的对应指针置空。

(4) 析构函数 $\sim\text{MAT}()$ `noexcept`:

如果 e 非空，直接使 r 及 c 置空即可。如果 e 指向了一段空间，需要将 e 所指向的空间进

行清理，然后将 r 与 c 置空。

(5) 成员函数矩阵加法 `MAT<T>::operator+(const MAT& a)const:`

如果两个矩阵不能相加，即双方行列数不同，或有一方矩阵失效，则抛出不能相加的异常。我们新建立一个结果矩阵 `res`，其长为 r 宽为 c ，并对其中每一个矩阵元素进行计算相加并赋值即可。

(6) 成员函数矩阵减法 `MAT<T>::operator-(const MAT& a)const:`

如果两个矩阵不能相减，即双方行列数不同，或有一方矩阵失效，则抛出不能相加的异常。我们新建立一个结果矩阵 `res`，其长为 r 宽为 c ，并对其中每一个矩阵元素进行计算相减并赋值即可。

(7) 成员函数矩阵乘法 `MAT<T>::operator*(const MAT& a)const:`

如果两个矩阵不能相乘，即前者的列数不等于后者的行数，或有一方矩阵失效，则抛出不能相加的异常。我们新建立一个结果矩阵 `res`，其长为 r 宽为 c ，并对其中每一个矩阵元素进行乘积运算并赋值即可。

(8) 成员函数矩阵转置 `MAT<T>::operator~()const:`

如果这个矩阵的行数或列数为 0，直接返回即可。否则建立一个结果矩阵并将行数和列数错置，并将每一个元素值赋值到对应位置上去。

(9) 操作符重载深拷贝赋值运算 `MAT<T>::operator=(const MAT& a):`

如果所赋值的 a 位置等于本地址，无需进行任何操作。否则，先调用本矩阵的析构函数，再在 `this` 指针的原地址上进行 a 的深拷贝构造，以建立一个新的对象。这种方法节约了空间，也使得代码简洁明快。

(10) 操作符重载移动赋值运算 `MAT<T>::operator=(MAT&& a)noexcept:`

如果所赋值的 a 位置等于本地址，无需进行任何操作。否则，如果我们的 e 指针指向了一段空间，应将其释放。之后进行指针的传递和置零操作即可。

(11) “`+=`” 运算符重载 `MAT<T>::operator+=(const MAT& a)`

返回 `*this = (*this) + a` 即可完成操作。

(12) “`-=`” 运算符重载 `MAT<T>::operator-=(const MAT& a)`

返回 `*this = (*this) - a` 即可完成操作。

(13) “`*=`” 运算符重载 `MAT<T>::operator*=(const MAT& a)`

返回 `*this = (*this) * a` 即可完成操作。

(14) 成员输出函数 `MAT<T>::print(char* s)const noexcept`

即将矩阵打印至字符串中即可。注意到模板 T 的类型，打印的参数不同，我们对 T 的类型进行了分类讨论和打印。

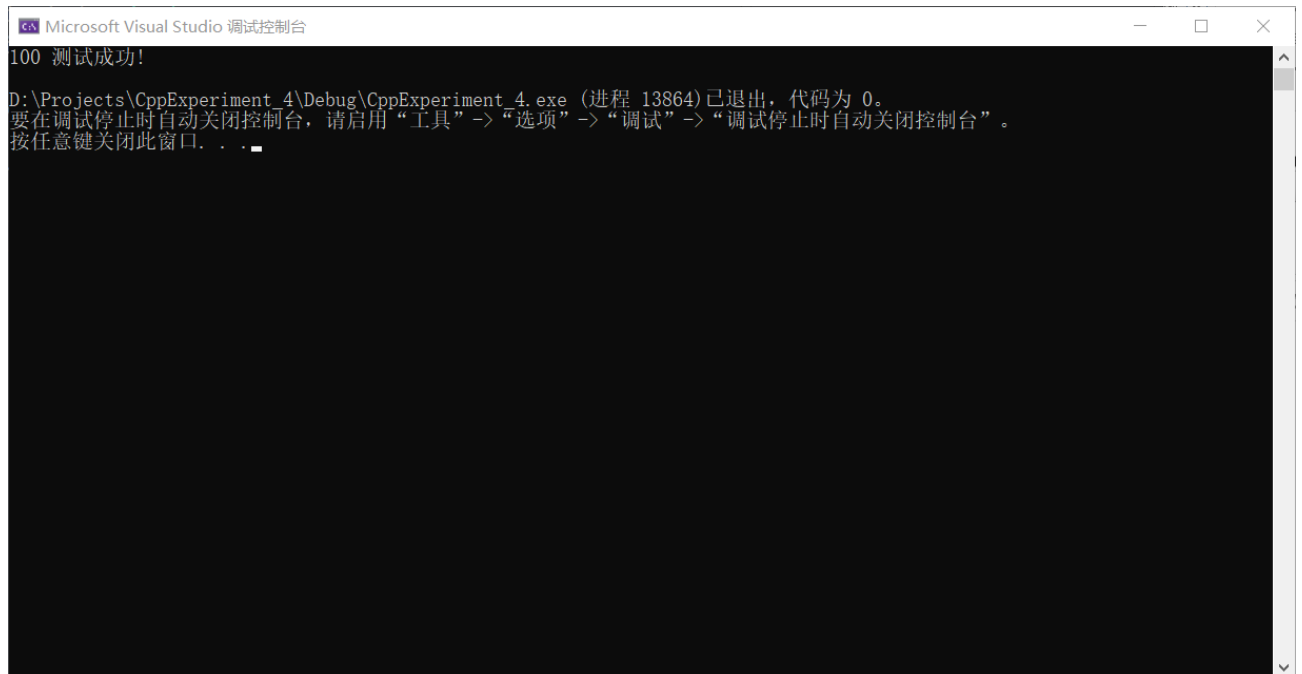
三、软件开发

本实验使用 Visual Studio Community 2022 进行开发、编译和连接生成可执行文件，Windows SDK 版本为 10.0.16299.0。使用 Visual Studio Community 2022 本地调试器进行的软

件调试。

四、软件测试

我们用实验课程所给出的测试库对源实验程序进行了测试，测试库通过了我们的程序并给出了 100 分。



为了涵盖几乎所有的功能，我们列出了我们自行的测试计划，并进行实地的检验，并输出相应的结果。

我们的测试计划为：

- (1) 建立长为 1 宽为 2 的矩阵 a 和 c，和一个长宽均为 2 的矩阵 b，并对 a 进行赋值。
- (2) 调用 `operator[]` 操作符重载函数对矩阵 b 进行赋值操作，查看输出结果。
- (3) 令 c 为矩阵 a 与 b 的乘积，并查看移动赋值结果。
- (4) 验证矩阵的加法。
- (5) 验证矩阵的减法。
- (6) 测试 “+=” 操作符重载结果。考虑到与 “-=” 和 “*=” 结构类似，只需验证这一个重载结果即可。
- (7) 测试矩阵的转置并输出。
- (8) 令 `c = c`，查看原地址的赋值情况。
- (9) 令 `d = c`，查看深拷贝构造的情况。
- (10) 令 `c = MAT<int>(2,2)`，查看移动赋值测试结果。

我们的测试代码附录如下。

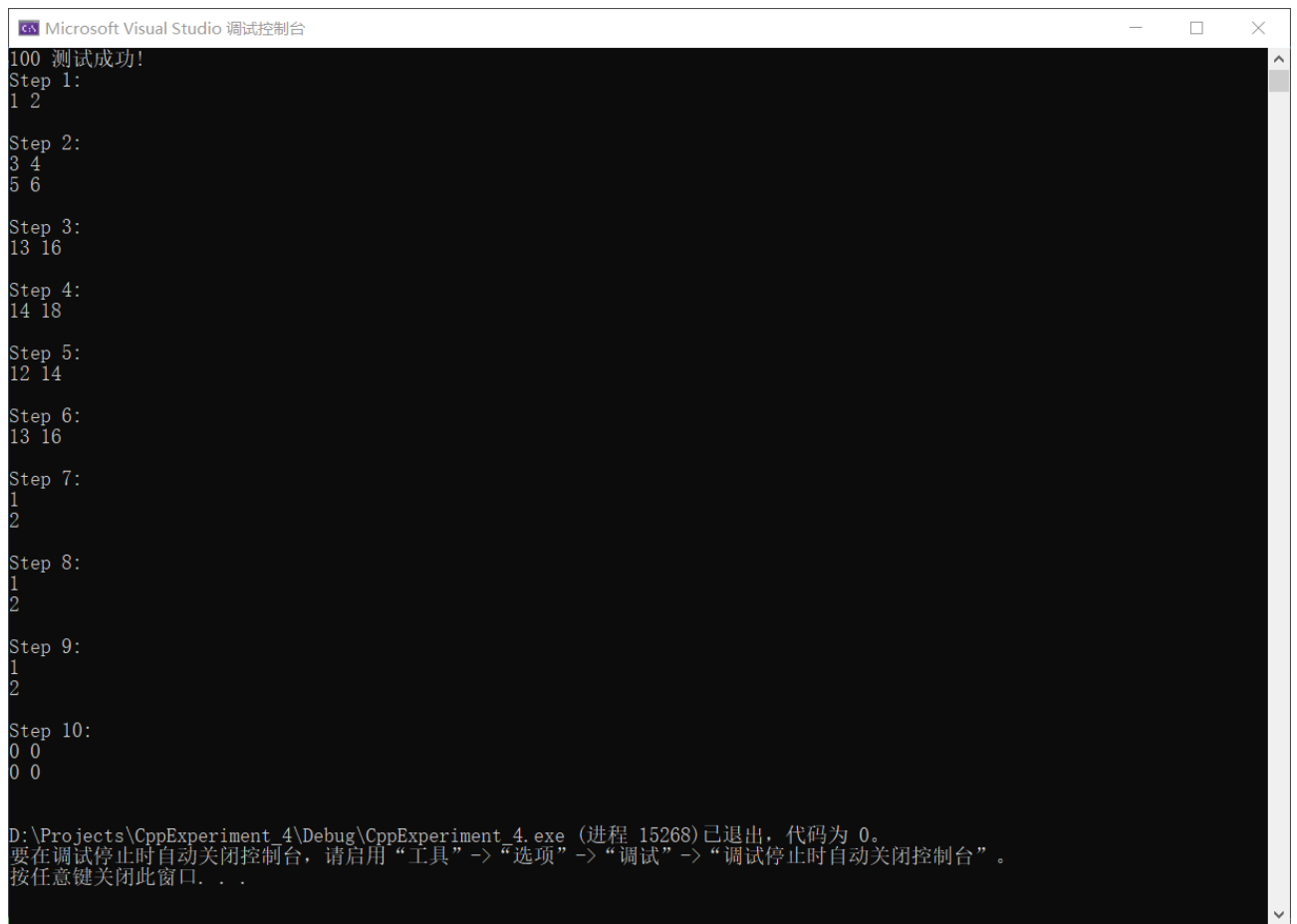
```
int s;  
const char* q = TestMAT(s);  
printf("%d %s \n", s, q);
```

```

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
printf("Step 1: \n%s\n", t);
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);
printf("Step 2: \n%s\n", t);
c = a * b;              //测试矩阵乘法运算
c.print(t);
printf("Step 3: \n%s\n", t);
(a + c).print(t);       //测试矩阵加法运算
printf("Step 4: \n%s\n", t);
c = c - a;              //测试矩阵减法运算
c.print(t);
printf("Step 5: \n%s\n", t);
c += a;                 //测试矩阵“+=”运算
c.print(t);
printf("Step 6: \n%s\n", t);
c = ~a;                 //测试矩阵转置运算
c.print(t);
printf("Step 7: \n%s\n", t);
c = c;                  // 原地址赋值情况测试
c.print(t);
printf("Step 8: \n%s\n", t);
MAT<int> d = c;          // 深拷贝构造测试
d.print(t);
printf("Step 9: \n%s\n", t);
c = MAT<int>(2, 2);      // 移动赋值测试
c.print(t);
printf("Step 10: \n%s\n", t);

```

我们的运行结果如下，可见程序正常运行。



```
Microsoft Visual Studio 调试控制台
100 测试成功!
Step 1:
1 2

Step 2:
3 4
5 6

Step 3:
13 16

Step 4:
14 18

Step 5:
12 14

Step 6:
13 16

Step 7:
1
2

Step 8:
1
2

Step 9:
1
2

Step 10:
0 0
0 0

D:\Projects\CppExperiment_4\Debug\CppExperiment_4.exe (进程 15268) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

五、特点与不足

1. 技术特点

使用面向对象的方式进行开发, 并且我们为逻辑层的站点类和线路类矩阵类进行了完备的构造、赋值的成员函数定义和运算符的重载, 不易抛出错误。

2. 不足和改进的建议

对于一个数学的矩阵, 我们未实现矩阵的求逆操作。这个操作实现较难, 具有一定挑战性, 可以考虑后续进行编写。

六、过程和体会

1. 遇到的主要问题和解决方法

这次实验中在深拷贝赋值时, 老师教了我一个方法, 尤其巧妙。这种方法节约了代码, 也节约了空间, 确实好。我们将原方法和改进方法附着如下。

`//深拷贝赋值运算`

```
template<typename T>
```

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

```

    if (this == &a) return *this;

    this->~MAT();

    new (this) MAT(a);

    //if (this->e) {
    //  delete[] this->e;
    //  *(T**)&e = NULL; // 删除后将 e 置零
    //}

    /*(int*)&c = a.c;
    /*(int*)&r = a.r;
    /*(T**)&e = new T[r * c];
    //for (int i = 0; i < a.r; ++i)
    //  for (int j = 0; j < a.c; ++j)
    //      ((MAT&)(*this))[i][j] = ((MAT&a)[i][j]);

    return *this;
}

```

注释部分为原先的代码。在原先的代码中，我们对每一个元素都进行了置空和重载。可是只需先调用 this 指针的析构函数，再在原地址上重建 MAT<T>的深拷贝构造即可。这种方法节约了空间。

2. 课程设计的体会

本次实验让我较好地实践到了面向对象程序设计方法来设计程序，体会到了一个矩阵类所需要定义的各种操作符重载函数及其它的成员函数。在完成实验的过程中我也学会了通过查阅文档来解决遇到的问题，这让我收获很多。

七、源码和说明

1. 文件清单及其功能说明

本实验文件存储在“..\ex4\”文件夹下，内含两个文件：实验报告（实验报告.doc）和源程序包（包括源代码及 Release 可执行文件，4.rar）

2. 用户使用说明书

只需打开对应文件的 Release 版本文件即可。建议将原代码放置如编译器进行重新编译，以使原代码正常运行。

3. 源代码

源代码附录如下。我们在每个文件开头部分添加了代码所属位置的注释信息，以便于代码的阅读。源代码已保存在附录的“4.rar”中以备进一步的查验。

```
/* in File ex4_MAT.h */

#pragma once

#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;

};

```

//矩阵定义

```

template<typename T>
MAT<T>::MAT(int r, int c) :e(new T[r * c]), r(r), c(c) {
    for (int i = 0; i < r * c; ++i)
        e[i] = 0;
}

```

//深拷贝构造

```

template<typename T>
MAT<T>::MAT(const MAT& a):r(a.r), c(a.c), e(new T[a.r * a.c]) {
    for (int i = 0; i < a.r * a.c; ++i)
        e[i] = a.e[i];
}

```

//移动构造

```

template<typename T>
MAT<T>::MAT(MAT&& a)noexcept:e(a.e),r(a.r),c(a.c) {
    *(T**)&a.e = NULL;
    *(int*)&a.c = 0;
    *(int*)&a.r = 0;
}

```

```

template<typename T>

```

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

```
    if (e) {
```

```
        delete[] e;
```

```
        *(T**)&e = nullptr;
```

```
    }
```

```
    *(int*)&c = 0;
```

```
    *(int*)&r = 0;
```

```
}
```

//取矩阵 r 行的第一个元素地址，r 越界抛异常

```
template<typename T>
```

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

```
    if (r < 0 || r >= this->r || e == NULL) throw "invalid number!";
```

```
    return e + r * this->c;
```

```
}
```

//矩阵加法，不能加抛异常

```
template<typename T>
```

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

```
    if (r != a.r || c != a.c) throw "can not add!";
```

```
    if (e == NULL || a.e == NULL) throw "can not add!";
```

// 如果行列数不同，则不能加减

```
MAT<T> res(r, c);
```

```
for (int i = 0; i < r; ++i)
```

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

```
        res[i][j] = ((MAT&)(*this))[i][j] + ((MAT&a)[i][j];
```

```
    }
```

```
return res;
```

```
}
```

//矩阵减法，不能减抛异常

template<typename T>

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

if (r != a.r || c != a.c) throw "can not minus!";

// 如果行列数不同，则不能加减

if (e == NULL || a.e == NULL) throw "can not add!";

MAT<T> res(r, c);

for (int i = 0; i < r; ++i)

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

res[i][j] = ((MAT&)(*this))[i][j] - ((MAT&a)[i][j];

}

return res;

}

//矩阵乘法，不能乘抛异常

template<typename T>

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

if (this->c != a.r) throw "can not multiple!";

if (e == NULL || a.e == NULL) throw "can not add!";

int r0 = this->r;

int c0 = a.c;

MAT<T> res(r0, c0); // 临时矩阵用以存储乘积结果

for (int i = 0; i < r0; ++i)

for (int j = 0; j < c0; ++j) {

// res[i][j] 即 第 i 行第 j 列乘积结果

res[i][j] = 0;

for (int k = 0; k < this->c; ++k)

res[i][j] += ((MAT&)(*this))[i][k] * ((MAT&a)[k][j];

}

return res;

```
}
```

//矩阵转置

```
template<typename T>
```

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

```
    if (r == 0 || c == 0) return *this; // 如果行列数为零，则无需转置
```

```
    int c0 = this->r;
```

```
    int r0 = this->c;
```

```
    MAT<T> res(r0, c0); // 临时矩阵用以存储转置结果
```

```
    for (int i = 0; i < r0; ++i)
```

```
        for (int j = 0; j < c0; ++j)
```

```
            res[i][j] = ((MAT&)(*this))[j][i];
```

```
    return res;
```

```
}
```

//深拷贝赋值运算

```
template<typename T>
```

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

```
    if (this == &a) return *this;
```

```
    this->~MAT();
```

```
    new (this) MAT(a);
```

```
    //if (this->e) {
```

```
        // delete[] this->e;
```

```
        // *(T**)e = NULL; // 删除后将 e 置零
```

```
    //}
```

```
    /*(int*)&c = a.c;
```

```
    /*(int*)&r = a.r;
```

```
    /*(T**)e = new T[r * c];
```

```
    //for (int i = 0; i < a.r; ++i)
```

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

```
//      ((MAT&)(*this))[i][j] = ((MAT&a)[i][j];
return *this;
}
```

//移动赋值运算

```
template<typename T>
MAT<T>& MAT<T>::operator=(MAT&& a)noexcept {
    if (this == &a) return *this;
    if (e) delete[]e;
    // 移动
    *(T**)&e = a.e;
    *(int*)&r = a.r;
    *(int*)&c = a.c;
    // 清零
    *(T**)&a.e = NULL;
    *(int*)&a.r = 0;
    *(int*)&a.c = 0;
    return *this;
}
```

//“+=”运算

```
template<typename T>
MAT<T>& MAT<T>::operator+=(const MAT& a) {
    return *this = *this + a;
}
```

//“-=”运算

```
template<typename T>
```

```

MAT<T>& MAT<T>::operator-=(const MAT& a) {
    return *this = *this - a;
}

//“*”运算
template<typename T>
MAT<T>& MAT<T>::operator*=(const MAT& a) {
    return *this = *this * a;
}

#define _CRT_SECURE_NO_WARNINGS
//print 输出至 s 并返回 s: 列用空格隔开, 行用回车结束
template<typename T>
char* MAT<T>::print(char* s) const noexcept {
    for (int i = 0; i < r; ++i)
    {
        for (int j = 0; j < c; ++j) {
            // 依次判断 T 的类型
            if (typeid(T) == typeid(long long)) s += sprintf(s, "%lld ",
((MAT&)*this)[i][j]);
            if (typeid(T) == typeid(int)) s += sprintf(s, "%d ", ((MAT&)*this)[i][j]);
            if (typeid(T) == typeid(double)) s += sprintf(s, "%lf ",
((MAT&)*this)[i][j]);
            if (typeid(T) == typeid(float)) s += sprintf(s, "%f ", ((MAT&)*this)[i][j]);
        }
        s += sprintf(s, "\n");
    }
    return s;
}

```

```
template MAT<int>;          //用于实验四，必须放在模板定义文件的尾部，用于强制实例化
template MAT<long long>;    //用于实验四，必须放在模板定义文件的尾部，用于强制实例化

/* in File ex4_test.cpp */

// CppExperiment_4.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "ex4_MAT.h"

extern const char* TestMAT(int& s); //用于实验四

int main(int argc, char* argv[])          //请扩展 main()测试其他运算
{
    int s;
    const char* q = TestMAT(s);
    printf("%d %s \n", s, q);
    //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
    //printf("%s\n", t);
    //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);
```



```
//printf("%s\n", t);  
//c = a * b;                //测试矩阵乘法运算  
//c.print(t);  
//printf("%s\n", t);  
//(a + c).print(t);          //测试矩阵加法运算  
//printf("%s\n", t);  
//c = c - a;                //测试矩阵减法运算  
//c.print(t);  
//printf("%s\n", t);  
//c += a;                   //测试矩阵“+=”运算  
//c.print(t);  
//printf("%s\n", t);  
//c = ~a;                   //测试矩阵转置运算  
//c.print(t);  
//printf("%s\n", t);  
return 0;  
}
```