⑪

華中科技大學

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 ：　计科 2011 班

学　　号 ：　U202015084

姓　　名 ：　　张文浩

指导教师 ：　　金良海

2021 年 12 月 16 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

本次实验要求用 C++实现对矩阵的基本操作，矩阵相关信息用类模板储存，矩阵元素可以是任意类型，矩阵类的基本操作包括初始化矩阵（用矩阵行数和列数初始化、用深拷贝方法初始化、用移动构造方法初始化）、取矩阵 r 行的第一个元素地址、矩阵加法、减法、乘法、转置、用深拷贝和移动构造方法实现赋值、“+=”“-=”“*=”运算、打印矩阵中的元素、销毁矩阵等功能。

二、系统设计

1. 概要设计

本次实验用类模板来表示矩阵，类的成员变量包括指向所有矩阵元素的指针、矩阵的行数和列数大小。所实现的对矩阵的操作包括初始化矩阵（用矩阵行数和列数初始化、用深拷贝方法初始化、用移动构造方法初始化）、取矩阵 r 行的第一个元素地址、矩阵加法、减法、乘法、转置、用深拷贝和移动构造方法实现赋值、“+=”“-=”“*=”运算、打印矩阵中的元素、销毁矩阵等。

本次实验采用专用测试库进行功能测试，没有人机交互界面。

所实现的函数功能模块主要包括：

```
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
```

总体流程图如下：

图 2-1 总体流程图

2. 详细设计

①矩阵类：

```
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;
};
```

②函数原型：MAT(int r, int c);

函数功能：用矩阵大小初始化矩阵

入口参数：矩阵行数 int r，矩阵列数 int c（隐含参数 this）

出口参数：无

流程：给 e 分配一个 r*c 大小的空间，将 r 赋值给 this->r，将 c 赋值给 this->c

③函数原型：MAT(const MAT& a)

函数功能：深拷贝构造矩阵

入口参数：已知矩阵引用 const MAT& a（隐含参数 this）

出口参数：无

流程：给 e 分配一个 a.r*a.c 大小的空间，将 a.r 赋值给 r，将 a.c 赋值给 c,遍历 e 与 a.e 使 e[i]=a.e[i]

④函数原型：MAT(MAT&& a)noexcept

函数功能：移动构造矩阵

入口参数：已知矩阵引用 MAT&& a（隐含参数 this）

出口参数：无

流程：将 a.e 赋值给 e，将 a.r 赋值给 r，将 a.c 赋值给 c，将 a 的所有成员变量赋值为 0

⑤函数原型：~MAT()noexcept

函数功能：析构矩阵

入口参数：无（隐含参数 this）

出口参数：无

流程：如果 e 非空，释放 e 的空间，将 e，r，c 赋值为 0

⑥函数原型：T* const operator[ ](int r)
函数功能：取矩阵 r 行的第一个元素地址，r 越界抛异常
入口参数：行数 int r（隐含参数 this）
出口参数：T 类型指针
流程：如果 r<0 或者 r 大于等于矩阵行数，抛出异常，否则返回 e+r*c

⑦函数原型：MAT operator+(const MAT& a)const
函数功能：矩阵加法，不能加抛异常
入口参数：已知矩阵引用 const MAT& a（隐含参数 this）
出口参数：相加所得矩阵
流程图：

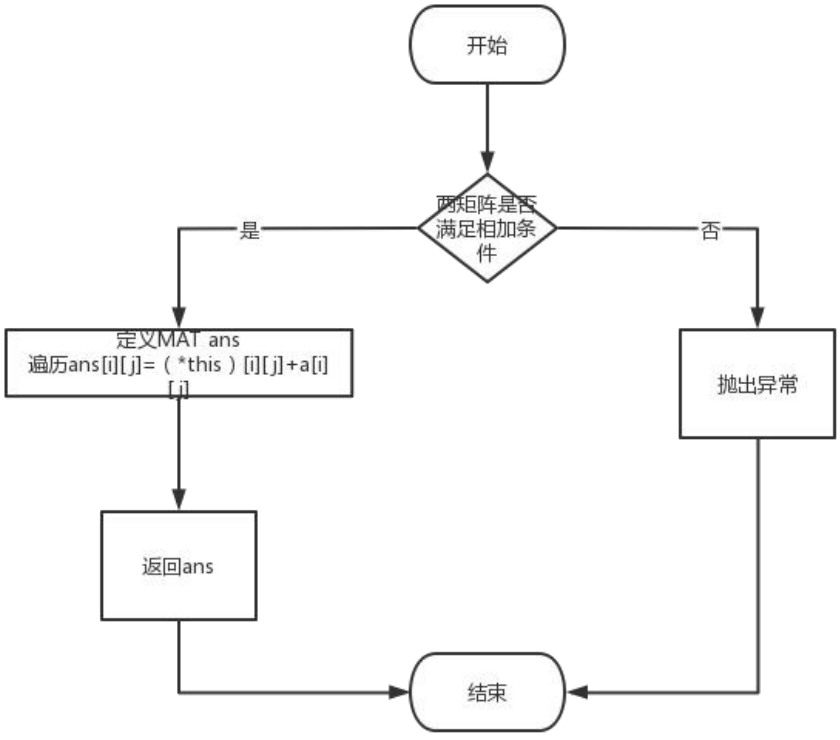

图 2-2 矩阵相加流程图

⑧函数原型：MAT operator-(const MAT& a)const
函数功能：矩阵减法，不能减抛异常
入口参数：已知矩阵引用 const MAT& a（隐含参数 this）
出口参数：相减所得矩阵
流程：与加法类似，只是将遍历时的加号改为减号

⑨函数原型：MAT operator*(const MAT& a)const

函数功能：矩阵乘法，不能乘抛异常

入口参数：已知矩阵引用 const MAT& a（隐含参数 this）

出口参数：相乘所得矩阵

流程图：

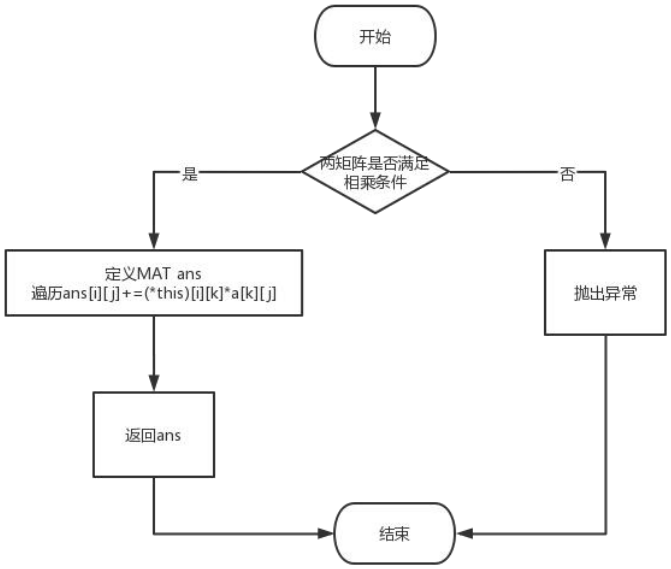

图 2-3　矩阵相乘流程图

⑩函数原型：MAT operator~()const

函数功能：矩阵转置

入口参数：无（隐含参数 this）

出口参数：转置所得矩阵

流程图：

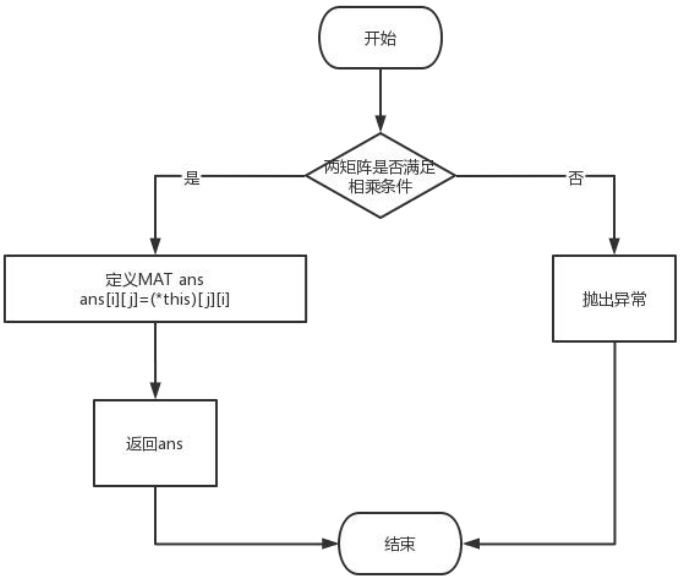

图 2-4　矩阵转置流程图

⑪函数原型：MAT& operator=(const MAT& a)
函数功能：深拷贝赋值运算
入口参数：已知矩阵引用 const MAT& a（隐含参数 this）
出口参数：赋值后的对象
流程：如果 this==&a，返回*this，如果 e 非空，释放空间并置零，之后与之前的深拷贝初始化相同。

⑫函数原型：MAT& operator=(MAT&& a)noexcept
函数功能：移动赋值运算
入口参数：已知矩阵引用 MAT&& a（隐含参数 this）
出口参数：赋值后的对象
流程：如果 this==&a，返回*this，如果 e 非空，释放空间并置零，之后与之前的移动初始化相同。

⑬函数原型：MAT& operator+=(const MAT& a)
函数功能：“+=”运算
入口参数：已知矩阵引用 const MAT& a（隐含参数 this）
出口参数：运算后的对象
流程：返回*this = *this + a

⑭函数原型：MAT& operator-=(const MAT& a)
函数功能：“-=”运算
入口参数：已知矩阵引用 const MAT& a（隐含参数 this）
出口参数：运算后的对象
流程：返回*this = *this - a

⑮函数原型：MAT& operator*=(const MAT& a)
函数功能：“*=”运算
入口参数：已知矩阵引用 const MAT& a（隐含参数 this）
出口参数：运算后的对象
流程：返回*this = *this * a

⑯函数原型：char* print(char* s)const noexcept
函数功能：print 输出至 s 并返回 s
入口参数：输出字符串 char* s
出口参数：输出字符串

流程：声明 stringstream 变量 ss，遍历每个元素，将元素写入 ss，利用 strcpy 将 ss.str().c_str()复制给 s，返回 s

三、软件开发

采用 VS2019 开发，编译模式为 Debug-X86 模式，利用本地 Windows 调试器进行调试。

四、软件测试

采用实验二测试库，测试结果如图 4-1 所示

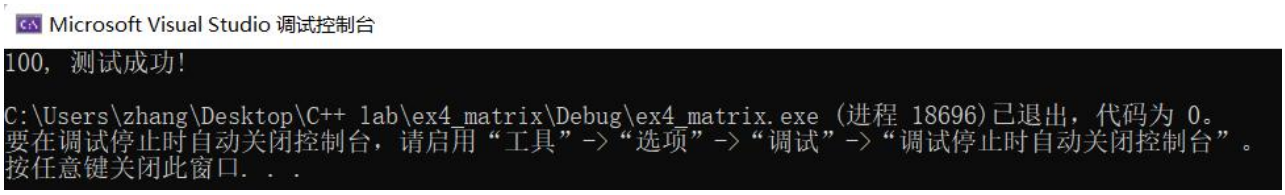

图 4-1　实验四测试图

五、特点与不足

1. 技术特点

使用类模板，可以定义任意元素类型的矩阵；合理利用已经重载的运算符，极大简化了代码。

2. 不足和改进的建议

对于类模板还不太熟悉，仅仅使用了一下模板的形式，一些具体知识仍然没有掌握。建议以后补齐这块知识漏洞。

对于一些函数（如 memset），不能正确使用，建议多看一些相关的资料。

对于 stringstream 不是很了解，应该查阅相关资料。

六、过程和体会

1. 遇到的主要问题和解决方法

在编写本次实验的时候还没有使用过模板，对于模板的形式和使用方法还不太熟悉，所以在编写程序之前首先阅读了相关的资料，也询问了身边的同学，才开始编写程序。

同时在自己刚开始编写的时候，我将.h 文件和.cpp 文件分开写，但编译时会报错，查阅资料发现因为在编译时模板并不能生成真正的二进制代码，而是在编译调用模板类或函数的.cpp 文件时才会去找对应的模板声明和实现，在这种情况下编译器是不知道实现模板类或函数的.cpp 文件的存在，所以它只能找到模板类或函数的声明而找不到实现，而只好创建一个符号寄希望于链接程序找地址。但模板类或函数的实现并不能被编译成二进制代码，结果链接程序找不到地址只好报错了。之后将类模板成员函数的实现也放进了.h 文件中，问题解决。

在编写矩阵相乘的函数时，首先定义了储存结果的对象 MAT ans，希望将 ans.e 的全部元素赋值为 0，第一遍编写的时候我使用了 memset 函数，但并没有通过测试，调试发现是 memset 并没有将 ans.e 的全部元素赋值为 0，应该是使用了模板的原因。之后我改用遍历将全部元素初始化为 0，通过测试。

2. 课程设计的体会

对于一些没有接触过的知识，用到了就需要学懂，而不能仅仅一知半解，仅仅是为了把这次实验完成，这样就根本没有起到补充学习的作用，特别是这次使用的类模板，是以后 C++ 编程中经常用到的，虽然没有纳入考试范围，但也应该把模板学好。

对于一些不太熟悉的函数，应该多查阅资料了解其使用条件再在程序中调用，否则就很可能会出现错误。

七、源码和说明

1. 文件清单及其功能说明

ex4_mat.h 是头文件（类声明和函数定义）

ex4_test.cpp 是源文件（main 函数）

ex4_matrix.sln 用来打开项目

“实验四测试库”文件夹中是本次实验所需要的测试库文件

其余文件为 vs 项目配置文件

2. 用户使用说明书

使用时，双击 ex4_matrix.sln 用来打开项目进入 vs 界面，将测试库添加进入工程，然后点击界面上方的“本地 Windows 调试器”即可运行程序。

3. 源代码

ex4_matrix.h:

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>
#include <exception>
#include <typeinfo>
#include <string.h>
#include <sstream>
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;                        //析构
    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<typename T>
MAT<T>::MAT(int r, int c) :e(new T[r*c]), r(r), c(c) {}//矩阵定义

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 < r * c; i++)
        e[i] = a.e[i];
}//深拷贝构造

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

```
    *(T **)&a.e = 0;
    *(int *)&a.r = 0;
    *(int *)&a.c = 0;
}//移动构造

template<typename T>
MAT<T>::~MAT()noexcept
{
    if (e)
    {
        delete[] e;
        *(T**)&e = 0;
        *(int*)&r = *(int*)&c = 0;
    }
}//析构函数

template<typename T>
T* const MAT<T>::operator[ ](int r)
{
    if (r<0 || r>=this->r)   //r 不符合要求
        throw("error");
    return e + r * c;   //数组名是第一个元素的地址，这是指针的移动
}//取矩阵 r 行的第一个元素地址，r 越界抛异常

template<typename T>
MAT<T> MAT<T>::operator+(const MAT& a)const
{
    if (r != a.r || c != a.c)   //是否可加
        throw("error");
    MAT ans(r, c);
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            ans[i][j] = ((MAT &)(*this))[i][j] + (*(MAT*)&a)[i][j];   //类型转换
    return ans;
}//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("error");
    MAT ans(r, c);
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            ans[i][j] = ((MAT&)(*this))[i][j] - (*(MAT*)&a)[i][j];
    return ans;
}//矩阵减法，不能减抛异常

template<typename T>
MAT<T> MAT<T>::operator*(const MAT& a)const
{
    if(c!=a.r)
        throw("error");
    MAT ans(r, a.c);
    for (int i = 0; i < r; i++)
        for (int j = 0; j < a.c; j++)
        {
            ans[i][j] = 0;     //初始化为 0 才能得到正确结果
            for (int k = 0; k < c; k++)
                ans[i][j] += ((MAT&)(*this))[i][k] * (*(MAT*)&a)[k][j];
        }
    return ans;
}//矩阵乘法，不能乘抛异常

template<typename T>
MAT<T> MAT<T>::operator~()const
{
    MAT ans(c, r);
    for (int i = 0; i < c; i++)
        for (int j = 0; j < r; j++)
            ans[i][j] = ((MAT&)(*this))[j][i];
    return ans;
```

```
}//矩阵转置

template<typename T>
MAT<T>& MAT<T>::operator=(const MAT& a)
{
    if (this == &a) //不能自己赋值给自己
        return *this;
    if (e) //e 非空才可以赋值
    {
        delete[] e;
        *(T**)&e = 0;
    }
    *(T**)&e = new T[a.r*a.c];
    *(int*)&r = a.r;
    *(int*)&c = a.c;
    for (int i = 0; i < r * c; i++)
        e[i] = a.e[i];
    return *this;
}//深拷贝赋值运算

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

```
    return *this;
}//移动赋值运算

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;
}//“*=”运算

template<typename T>
char* MAT<T>::print(char* s)const noexcept
{
    stringstream ss;
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            ss << ((MAT&)(*this))[i][j]<<",";
        ss << '\n';
    }
    string str = ss.str();
    strcpy(s, str.c_str());
    return s;
}//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>;
```

ex4_test.cpp:

```
#include "ex4_mat.h"
extern const char* TestMAT(int& s);   //测试函数
int main(int argc, char* argv[])
{
    int s;   //分数
    const char* q = TestMAT(s);   //提示信息
    printf("%d, %s\n", s, q);
    return 0;
}
```