華中科技大學

课 程 实 验 报 告

课程名称：面向对象程序设计

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 ：**CS2008 班**

学　　号 ：**U202015533**

姓　　名 ：徐瑞达

指导教师 ：许向文

2021 年 **10** 月 **26** 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。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

1. 声明模板类 MAT，并实现其成员函数的定义。
2. 使用矩阵定义、深拷贝构造、移动构造三种方式重载构造函数，其中移动构造函数需要将参数 a 置空；
3. 重载矩阵的加法、减法、乘法运算符，当无法进行运算时抛出异常；
4. 定义深拷贝赋值和移动赋值运算，其中移动构造时需要将参数 a 赋空；
5. 重载矩阵的转置、自加、自减、自乘运算符；
6. 将矩阵格式化输出至 s，并要根据实例类型使用不同的格式化输出方式；
7. 在 main 函数中可以使用 TestMAT 函数进行测试，也应可以自己调用函数测试；
8. 至少实现两种实例类，本项目实现了四种实例类——int、long long、float、double。

二、系统设计

1. 概要设计

根据用户需求，将该项目系统按照函数功能分为若干个模块进行时实现。将 MAT 类声明和定义封装在 MAT 头文件中，在 main 源文件中调用 TestMAT 函数测试或者自写程序测试，用户可根据提示进行选择。

总体结构流程图如下：

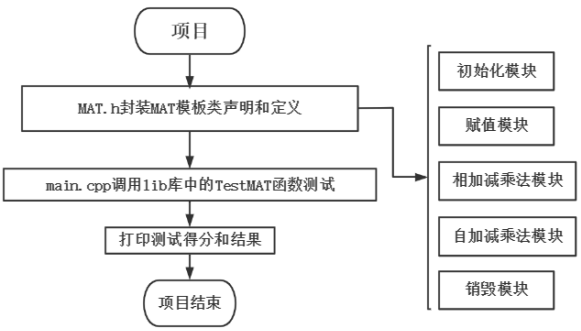

图 1 总体结构流程图

2. 详细设计

1. 初始化模板类模块：

该模块使用函数重载定义了三个构造函数，分别实现根据参数（行和列）初始化，深拷贝初始化，移动初始化三个功能。其中 MAT(int r, int c)函数根据参数创建一个行数为 r、列数为 c 的矩阵；MAT(const MAT &a)函数将模板类 a 深拷贝至 this 所指类；MAT(MAT &&a) noexcept 函数用 a 移动初始化 this 所指类，并要将 a 赋空。具体流程图如图 2 所示。

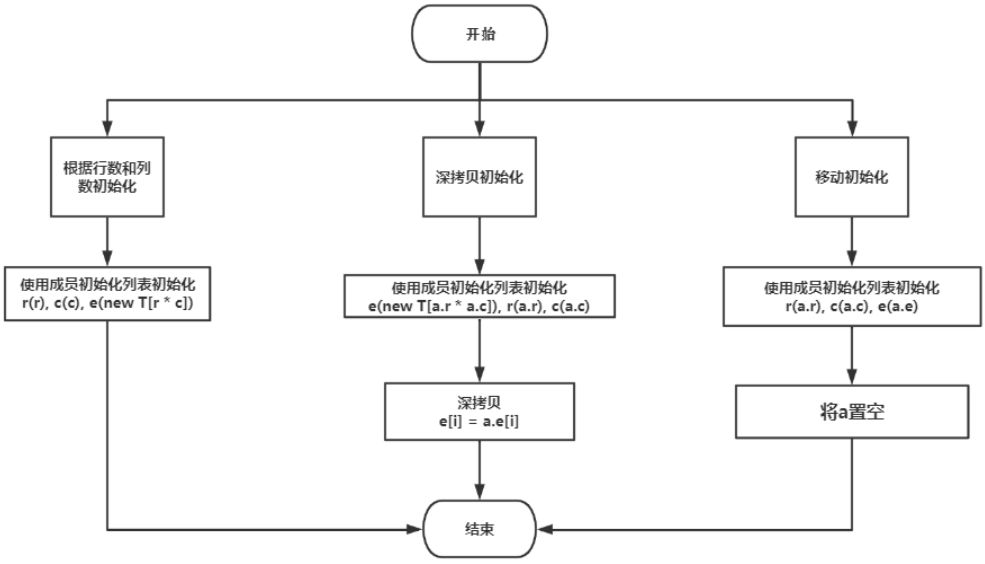

图 2 初始化模块流程图

2. 赋值模块：

该模块使用函数重载定义了两个运算符'='重载函数，分别实现深拷贝赋值，移动赋值两个功能。其中 virtual MAT &operator=(const MAT &a)函数将模板类 a 深拷贝至 this 所指类；virtual MAT &operator=(MAT &&a) noexcept 函数用 a 移动赋值 this 指类，并要将 a 赋空。具体流程图如图 3 所示。

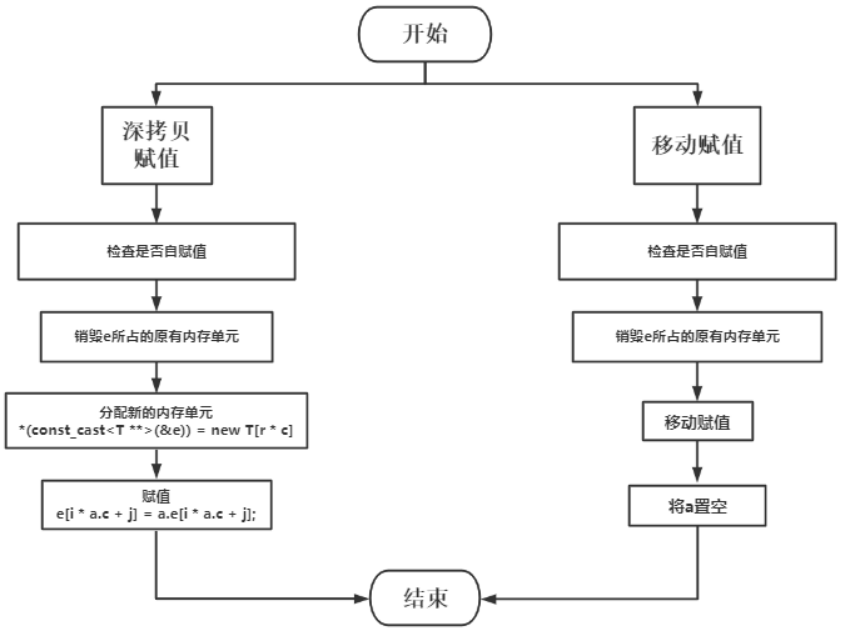

图 3 赋值模块流程图

3. 相加减乘法模块

加法和减法操作中需要判断行数和列数是否一致；乘法需要判断该矩阵的行数和参数 a 的列数是否相同，当不满足以上条件时应抛出对应的异常信息。其中 virtual MAT operator+(const MAT &a) const 进行矩阵加法；virtual MAT operator-(const MAT &a) const 进行矩阵减法；virtual MAT operator*(const MAT &a) const 进行矩阵乘法。具体流程图如图 4 所示。

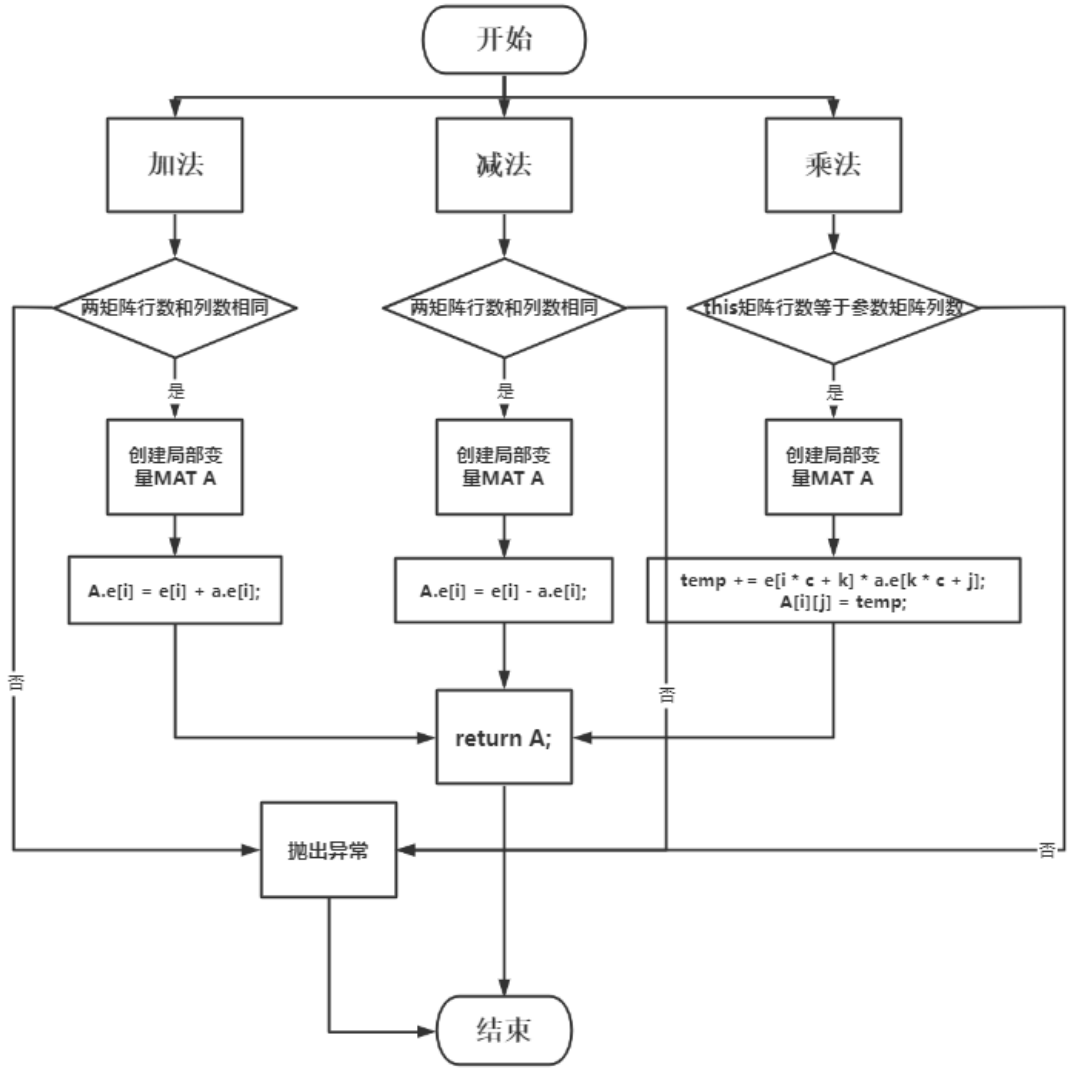

图 4 相加减乘法模块流程图

4. 自加减乘法模块：

自加和自减操作中需要判断行数和列数是否一致；自乘需要判断该矩阵的行数和参数 a 的列数是否相同，当不满足以上条件时应抛出对应的异常信息。这类运算和相加减乘法运算返回 MAT 副本不同，直接返回 MAT 引用，并且是在该 MAT 类型上直接进行运算操作。其中 virtual MAT &operator+=(const MAT &a)进行矩阵加法；virtual MAT &operator-=(const MAT &a)进行矩阵减法；virtual MAT &operator*=(const MAT &a)进行矩阵乘法。具体流程图如图 5 所示。

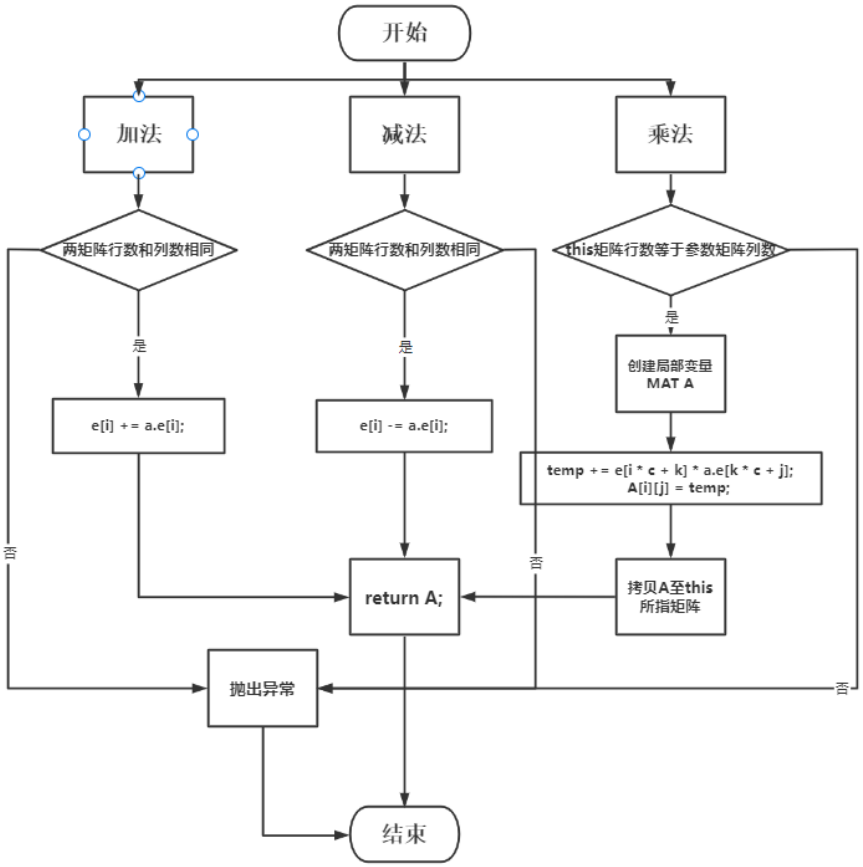

图 5 自加减乘法模块流程图

5. 销毁模块：销毁类时，使用 delete 删除分配的内存空间，并将各成员变量置空。具体函数功能实现方式可参阅附录源代码。

三、软件开发

采用 Windows11 系统下 Microsoft Visual Stdio 2019 平台进行开发，使用 MSVC 编译器进行编译链接生成解决方案与可执行文件，使用面向 MSVC 编译器的 Windows X86-Debug 调试器进行调试。

四、软件测试

使用 TestMAT 函数对程序进行测试，在调试过程中得到返回值如下图所示：

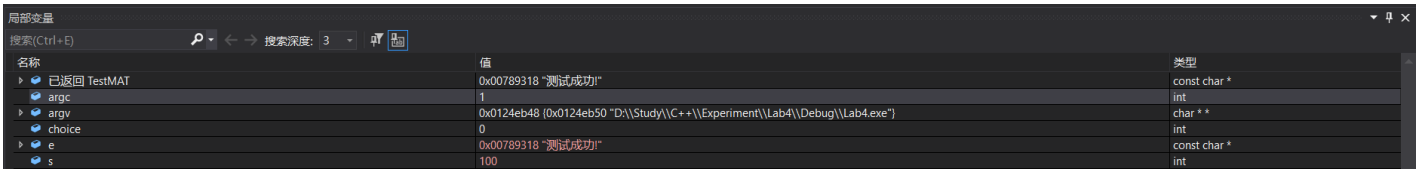

图 6 调试过程结果

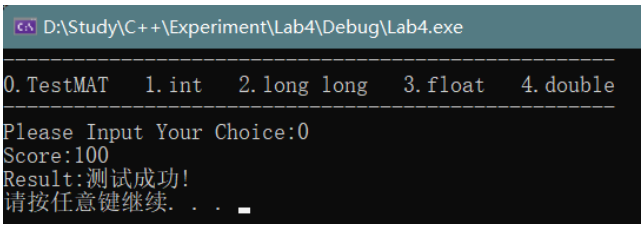

图 7 运行窗口结果一

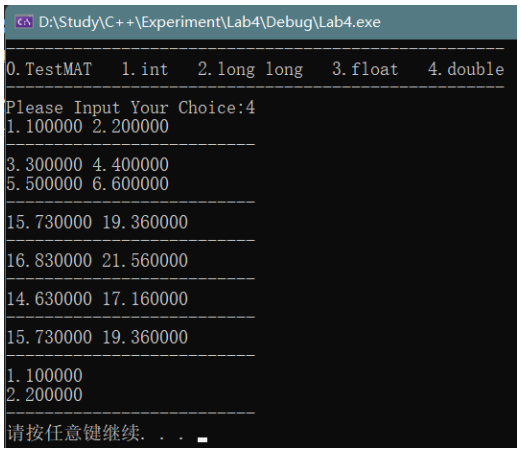

图 7 运行窗口结果二（选择 double 实例类）

五、特点与不足

1. 技术特点

（1） 用户可选择测试四种实例类或使用 Test 库函数

（2） 成员函数的定义简洁易懂，逻辑清晰。

2. 不足和改进的建议

代码中可能仍存在一些情况尚未考虑，在一些函数定义中存在代码不够精简，不尽易懂的情况。另外，对于基类和派生类的理解尚浅也让代码中一些东西冗余，需要改进。

六、过程和体会

1. 遇到的主要问题和解决方法

（1） 最初对模板类和实例类相关代码不甚明晰，导致在书写代码时总是有错误信息，后来在仔细阅读教材内容后，规避了种种错因；

（2） 在书写有关加减乘法运算的函数时，相加减乘法函数的返回值是 MAT 副本，而自加减乘法函数的返回值是 MAT 引用，由于未弄清楚而导致错误，最终在重新梳理成员函数声明后才得以正确实现定义。

2. 课程设计的体会

通过这次实验，我对模板类和实例类有了更为深入的理解，并认识到了模板类的强大之

处，而在这次实验中，我也对 noexcept 关键字有了进一步的了解。

七、源码和说明

1. 文件清单及其功能说明

ex4/project 文件夹中包含项目文件、头文件 MAT.h 和源文件 main.cpp,子文件夹 Release/ex4.exe 为可执行文件。

2. 用户使用说明书

打开文件夹后可直接运行 Release/Lab4.exe 程序，也可使用 Visual Stdio 2019 打开工程文件进行运行或调试。

3. 源代码

```
----------------------------MAT.h--------------------------------
#ifndef _MAT_H
#define _MAT_H
#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
#include <cstring>
#include <exception>
#include <iomanip>
#include <iostream>
#include <typeinfo>
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 <typename T>
MAT<T>::MAT(int r, int c) : r(r), c(c), e(new T[r * 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) //深拷贝构造
{
 for (int i = 0; i < a.r * a.c; i++)
       e[i] = a.e[i];
}
template <typename T>
MAT<T>::MAT(MAT &&a) noexcept : r(a.r), c(a.c), e(a.e) //移动构造
{
 *(const_cast<int *>(&a.r)) = 0;
 *(const_cast<int *>(&a.c)) = 0;
 *(const_cast<T **>(&a.e)) = nullptr;
}
template <typename T>
MAT<T>::~MAT() noexcept
{
 if (e!=nullptr)
       delete[] e;
 *(const_cast<int *>(&r)) = 0;
 *(const_cast<int *>(&c)) = 0;
 *(const_cast<T **>(&e)) = nullptr;
```

```
}
template <typename T>
T *const MAT<T>::operator[](int r) //取矩阵 r 行的第一个元素地址，r 越界抛异常
{
 if (r < this->r && r >= 0)
      return &e[r * c];
 else
      throw "Operator[] ERROR!";
}
template <typename T>
MAT<T> MAT<T>::operator+(const MAT &a) const //矩阵加法，不能加抛异常
{
 if (r == a.r && c == a.c)
 {
      MAT A(r, c);
      for (int i = 0; i < r * c; i++)
      {
           A.e[i] = e[i] + a.e[i];
      }
      return A;
 }
 else
      throw("Operator+ ERROR!");
}
template <typename T>
MAT<T> MAT<T>::operator-(const MAT &a) const //矩阵减法，不能减抛异常
{
 if (r == a.r && c == a.c)
 {
      MAT A(r, c);
      for (int i = 0; i < r * c; i++)
      {
           A.e[i] = e[i] - a.e[i];
      }
      return A;
 }
```

```
 else
        throw("Operator- ERROR!");
}
template <typename T>
MAT<T> MAT<T>::operator*(const MAT &a) const //矩阵乘法，不能乘抛异常
{
 if (c == a.r)
 {
        MAT A(r, a.c);
        for (int i = 0; i < r; i++)
        {
              for (int j = 0; j < a.c; j++)
              {
                    T temp = 0;
                    for (int k = 0; k < c; k++)
                          temp += e[i * c + k] * a.e[k * c + j];
                    A[i][j] = temp;
              }
        }
        return A;
 }
 else
        throw("Operator* ERROR!");
}
template <typename T>
MAT<T> MAT<T>::operator~() const //矩阵转置
{
 MAT A(c, r);
 for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
              A[j][i] = e[i * c + j];
 return A;
}
template <typename T>
MAT<T> &MAT<T>::operator=(const MAT &a) //深拷贝赋值运算
{
```

```
 if (this != &a)
 {
      if (e != nullptr)
           delete[] e;
      *(const_cast<int *>(&r)) = a.r;
      *(const_cast<int *>(&c)) = a.c;
      *(const_cast<T **>(&e)) = new T[r * c];
      for (int i = 0; i < a.r; i++)
           for (int j = 0; j < a.c; j++)
                e[i * a.c + j] = a.e[i * a.c + j];
 }
 return *this;
}
template <typename T>
MAT<T> &MAT<T>::operator=(MAT &&a) noexcept //移动赋值运算
{
 if (this != &a)
 {
      if (e != nullptr)
           delete[] e;
      *(const_cast<int *>(&r)) = a.r;
      *(const_cast<int *>(&c)) = a.c;
      *(const_cast<T **>(&e)) = a.e;
      *(const_cast<int *>(&a.r)) = 0;
      *(const_cast<int *>(&a.c)) = 0;
      *(const_cast<T **>(&a.e)) = nullptr;
 }
 return *this;
}
template <typename T>
MAT<T> &MAT<T>::operator+=(const MAT &a) //“+=”运算
{
 if (r == a.r && c == a.c)
 {
      for (int i = 0; i < r * c; i++)
      {
```

```
            e[i] += a.e[i];
        }
    }
    else
        throw("Operator+ ERROR!");
    return *this;
}
template <typename T>
MAT<T> &MAT<T>::operator-=(const MAT &a) //“-=”运算
{
    if (r == a.r && c == a.c)
    {
        for (int i = 0; i < r * c; i++)
        {
            e[i] -= a.e[i];
        }
    }
    else
        throw("Operator- ERROR!");
    return *this;
}
template <typename T>
MAT<T> &MAT<T>::operator*=(const MAT &a) //“*=”运算
{
    if (c == a.r)
    {
        MAT A(r, a.c);
        for (int i = 0; i < r; i++)
        {
            for (int j = 0; j < a.c; j++)
            {
                T temp = 0;
                for (int k = 0; k < c; k++)
                    temp += e[i * c + k] * a.e[k * c + j];
                A[i][j] = temp;
            }
```

```
        }
        delete[] e;
        *(const_cast<T **>(&e)) = new T[r * a.c];
        for (int i = 0; i < r; i++)
            for (int j = 0; j < a.c; j++)
                e[i * a.c + j] = A[i][j];
    }
    else
        throw("Operator* ERROR!");
    return *this;
}
template <typename T>
char *MAT<T>::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
{
    s[0] = '\0';
    if (typeid(e[0]) == typeid(int))
    {
        for (int i = 0; i < r; i++)
        {
            for (int j = 0; j < c; j++)
            {
                if (j != 0)
                {
                    char *temp = s + strlen(s);
                    sprintf(temp, "%c", ' ');
                }
                char *temp = s + strlen(s);
                sprintf(temp, "%6ld", (int)e[i * c + j]);
                if (j == c - 1)
                {
                    char *temp = s + strlen(s);
                    sprintf(temp, "%c", '\n');
                }
            }
        }
```

```
}
else if (typeid(e[0]) == typeid(long long))
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
        {
            if (j != 0)
            {
                char *temp = s + strlen(s);
                sprintf(temp, "%c", ' ');
            }
            char *temp = s + strlen(s);
            sprintf(temp, "%6lld", (long long)e[i * c + j]);
            if (j == c - 1)
            {
                char *temp = s + strlen(s);
                sprintf(temp, "%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)
            {
                char *temp = s + strlen(s);
                sprintf(temp, "%c", ' ');
            }
            char *temp = s + strlen(s);
            sprintf(temp, "%8f", (float)e[i * c + j]);
            if (j == c - 1)
```

```
                {
                    char *temp = s + strlen(s);
                    sprintf(temp, "%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)
                {
                    char *temp = s + strlen(s);
                    sprintf(temp, "%c", ' ');
                }
                char *temp = s + strlen(s);
                sprintf(temp, "%8lf", (double)e[i * c + j]);
                if (j == c - 1)
                {
                    char *temp = s + strlen(s);
                    sprintf(temp, "%c", '\n');
                }
            }
        }
    }
    return s;
}

#endif
```

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

```
#include "MAT.h"
template MAT<int>;
template MAT<long long>;
```

```
template MAT<float>;
template MAT<double>;
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
{
 cout << "----------------------------------------------------" << endl;
 cout << "0.TestMAT      1.int      2.long long      3.float      4.double" << endl;
 cout << "----------------------------------------------------" << endl;
 cout << "Please Input Your Choice:";
 int choice;
 cin >> choice;
 if (choice == 0)
 {
      int s;
      const char *e = TestMAT(s);
      cout << "Score:" << s << "\nResult:" << e << endl;
 }
 else if (choice == 1)
 {
      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
      cout << t << "-------------------------" << endl;
      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);
      cout << t << "-------------------------" << endl;
      c = a * b; //测试矩阵乘法运算
      c.print(t);
      cout << t << "-------------------------" << endl;
      (a + c).print(t); //测试矩阵加法运算
      cout << t << "-------------------------" << endl;
```

```
        c = c - a; //测试矩阵减法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c += a; //测试矩阵“+=”运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c = ~a; //测试矩阵转置运算
        c.print(t);
        cout << t << "-------------------------" << endl;
    }
    else if (choice == 2)
    {
        MAT<long long> a(1, 2), b(2, 2), c(1, 2);
        char t[2048];
        a[0][0] = 1ll; //类似地初始化矩阵的所有元素
        a[0][1] = 2ll; //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
        cout << t << "-------------------------" << endl;
        b[0][0] = 3ll;
        b[0][1] = 4ll; //调用 T* const operator[ ](int r)初始化数组元素
        b[1][0] = 5ll;
        b[1][1] = 6ll;
        b.print(t);
        cout << t << "-------------------------" << endl;
        c = a * b; //测试矩阵乘法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        (a + c).print(t); //测试矩阵加法运算
        cout << t << "-------------------------" << endl;
        c = c - a; //测试矩阵减法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c += a; //测试矩阵“+=”运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c = ~a; //测试矩阵转置运算
```

```
        c.print(t);
        cout << t << "-------------------------" << endl;
    }
    else if (choice == 3)
    {
        MAT<float> a(1, 2), b(2, 2), c(1, 2);
        char t[2048];
        a[0][0] = 1.1f; //类似地初始化矩阵的所有元素
        a[0][1] = 2.2f; //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
        cout << t << "-------------------------" << endl;
        b[0][0] = 3.3f;
        b[0][1] = 4.4f; //调用 T* const operator[ ](int r)初始化数组元素
        b[1][0] = 5.5f;
        b[1][1] = 6.6f;
        b.print(t);
        cout << t << "-------------------------" << endl;
        c = a * b; //测试矩阵乘法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        (a + c).print(t); //测试矩阵加法运算
        cout << t << "-------------------------" << endl;
        c = c - a; //测试矩阵减法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c += a; //测试矩阵“+=”运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c = ~a; //测试矩阵转置运算
        c.print(t);
        cout << t << "-------------------------" << endl;
    }
    else if (choice == 4)
    {
        MAT<double> a(1, 2), b(2, 2), c(1, 2);
        char t[2048];
```

```
        a[0][0] = 1.1; //类似地初始化矩阵的所有元素
        a[0][1] = 2.2; //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
        cout << t << "-------------------------" << endl;
        b[0][0] = 3.3;
        b[0][1] = 4.4; //调用 T* const operator[ ](int r)初始化数组元素
        b[1][0] = 5.5;
        b[1][1] = 6.6;
        b.print(t);
        cout << t << "-------------------------" << endl;
        c = a * b; //测试矩阵乘法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        (a + c).print(t); //测试矩阵加法运算
        cout << t << "-------------------------" << endl;
        c = c - a; //测试矩阵减法运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c += a; //测试矩阵“+=”运算
        c.print(t);
        cout << t << "-------------------------" << endl;
        c = ~a; //测试矩阵转置运算
        c.print(t);
        cout << t << "-------------------------" << endl;
    }
    system("pause");
    return 0;
}
```