C++ 语 言 程 序 设 计

实
验
报
告

实 验 三

姓名：方 尧

学号：190410102

班级：19 自动化 1 班

一 实验项目

1. 编写矩阵类 Matrix_4x4，实现矩阵初始化、求逆、转置、访问等基本功能
2. 基于运算符重载，实现矩阵的加减乘、输入输出的操作
3. 具体内容如下：

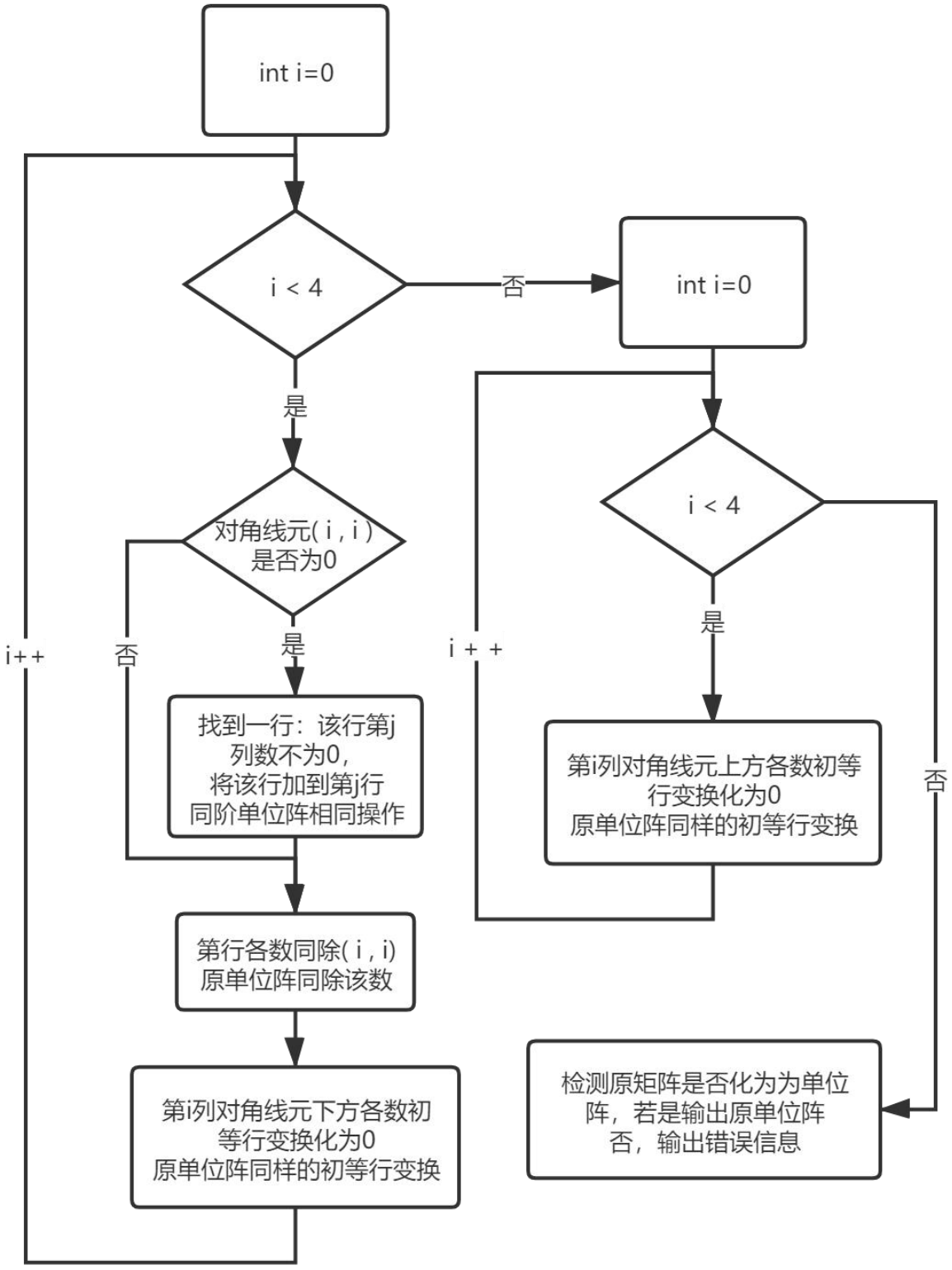
```
class Matrix_4x4
{
private:
    double matrix[4][4];
public:
    //默认构造函数，初始化矩阵为单位阵
    //带参数构造函数，用一个 4x4 的二维数组初始化
    //拷贝构造函数
    //重载 = 运算符，参数为矩阵对象
    //重载 = 运算符，参数为一个 4x4 的二维数组
    // 重载算术运算符 + - *
    // 重载 ^ 运算符为矩阵的 i 次幂（如果 i 为负数，如何处理？）
    // 重载 [ ] 运算符，实现双下标方式访问矩阵元素（该功能已经实现，无需自己写）
    const double * operator[] (const int i) const {return matrix[i];}
          double * operator[] (const int i)       {return matrix[i];}
   // 重载输入<< 和输出 >>
    Matrix_4x4 inverse();    //矩阵求逆，不改变当前矩阵值，返回逆矩阵
    Matrix_4x4 transpose(); //矩阵转置，不改变当前矩阵值，返回转置矩阵
};
```

二 实验原理

1、 说明矩阵求逆与行列式的原理与操作方法

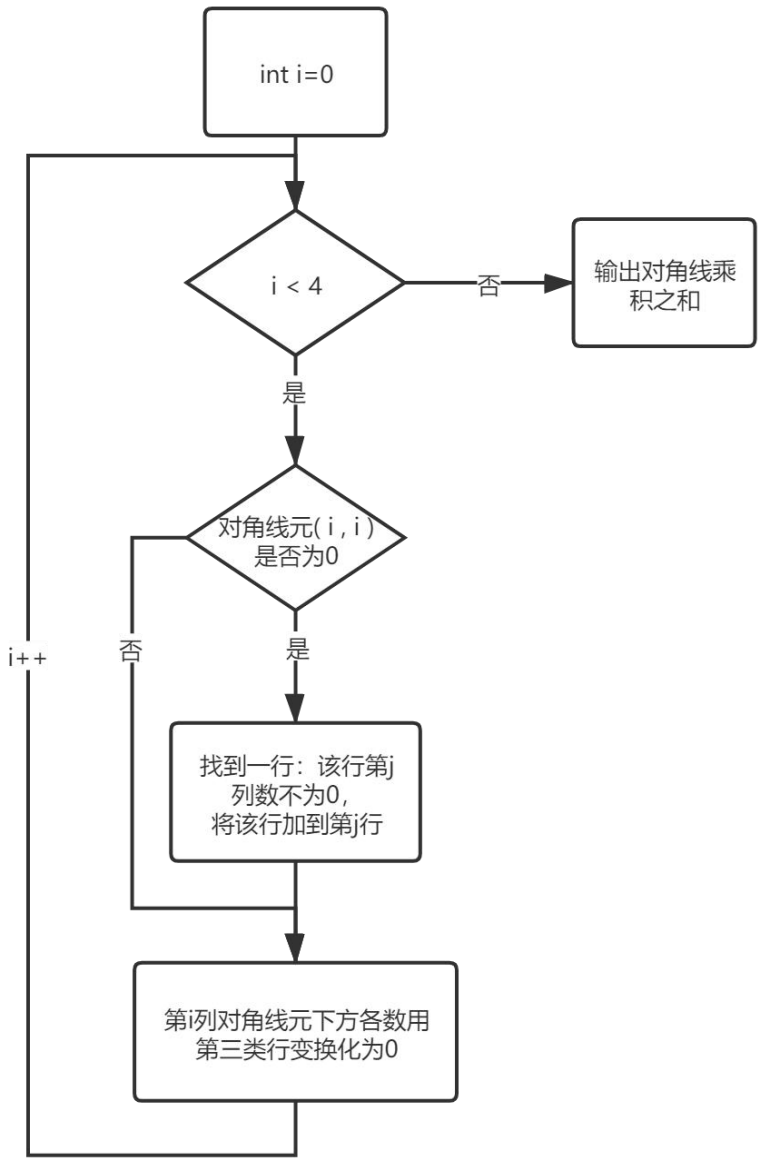
1) 求逆矩阵：

用初等行变换求逆矩阵



2) 求行列式

用第三列初等行变换即可求行列式



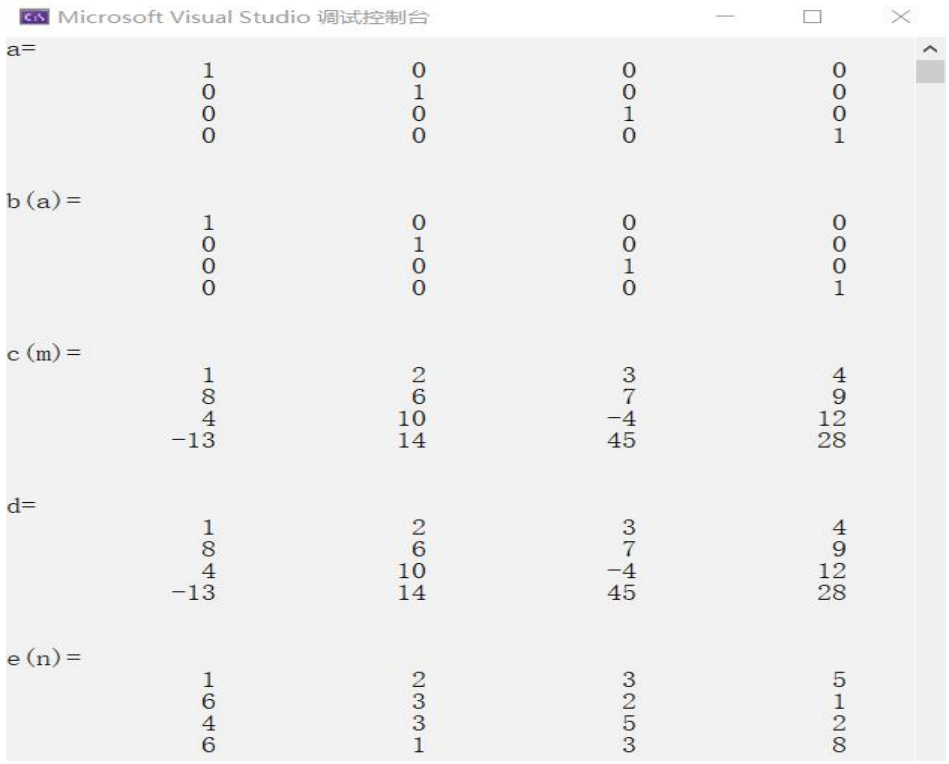
2、 矩阵类程序实现与结果（给出算法的源代码，说明关键代码的操作含义，给出运行结果）

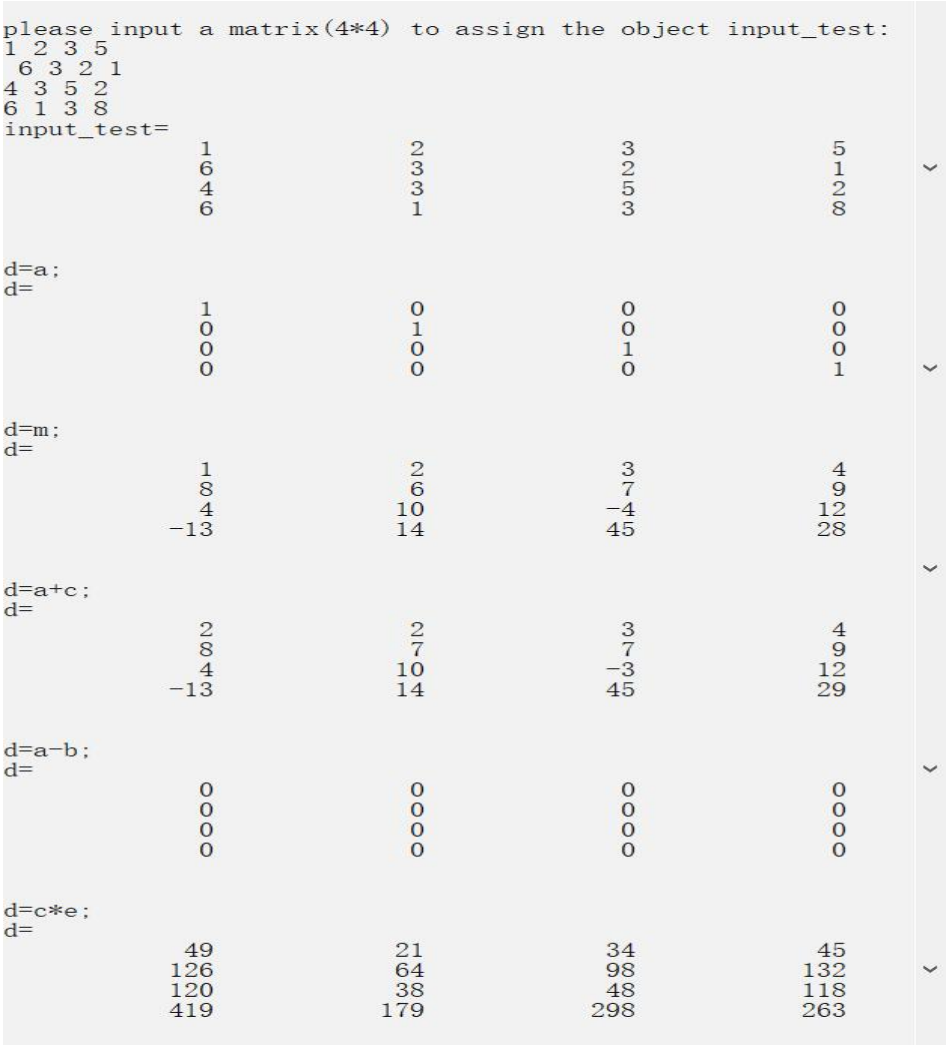
1) 源代码见附录

2) 关键代码

```
Matrix_4x4 inverse();//求逆函数
//矩阵 line2 行加到矩阵 line1 行;重载达到指定行对角线元化为 1
void add_multiply(Matrix_4x4& a, int line1 = -1, int line2 = -1);
//行变换将(line，col)位置的元置为 0
void to0(Matrix_4x4& a, int line, int col);
```

3) 运行结果截图


Microsoft Visual Studio 调试控制台
a=
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
b(a)=
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
c(m)=
1 2 3 4
8 6 7 9
4 10 -4 12
-13 14 45 28
d=
1 2 3 4
8 6 7 9
4 10 -4 12
-13 14 45 28
e(n)=
1 2 3 5
6 3 2 1
4 3 5 2
6 1 3 8


please input a matrix(4*4) to assign the object input_test:
1 2 3 5
6 3 2 1
4 3 5 2
6 1 3 8
input_test=
1 2 3 5
6 3 2 1
4 3 5 2
6 1 3 8
d=a;
d=
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
d=m;
d=
1 2 3 4
8 6 7 9
4 10 -4 12
-13 14 45 28
d=a+c;
d=
2 2 3 4
8 7 7 9
4 10 -3 12
-13 14 45 29
d=a-b;
d=
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
d=c*e;
d=
49 21 34 45
126 64 98 132
120 38 48 118
419 179 298 263

```
d =e^-3;
d=
  -0.015263109   0.016813543  -0.006218071   0.0046349695
    0.11987597     0.1484321   -0.11891857   -0.074095433
   -0.10760065   -0.16458853    0.13623971    0.064248793
    0.03737376   0.037196604  -0.038802321   -0.016809774


d =e^-2;
d=
   0.088555684  0.0058951685  -0.029347687  -0.034858388
    0.09022171     0.2541971   -0.16038703   -0.082788671
   -0.16468025   -0.23599897    0.22196591    0.083877996
  0.0044854543   0.053120595  -0.057926439    0.011982571


d =e^-1;
d=
   -0.20261438   0.071895425   0.026143791    0.11111111
    0.47712418    0.37908497   -0.27124183   -0.27777778
   -0.18954248    -0.2875817    0.39542484    0.055555556
    0.16339869  0.0065359477   -0.13398693    0.055555556


d =e^0;
d=
             1             0             0             0
             0             1             0             0
             0             0             1             0
             0             0             0             1


d =e^1;
d=
             1             2             3             5
             6             3             2             1
             4             3             5             2
             6             1             3             8


d =e^2;
d=
            55            22            37            53
            38            28            37            45
            54            34            49            49
            72            32            59           101


d =e^3;
d=
           653           340           553           795
           624           316           490           652
           748           406           622           794
          1106           518           878          1318
```

```
d=e.transpose();
d=
             1             6             4             6
             2             3             3             1
             3             2             5             3
             5             1             2             8


d=e.inverse();
d=
   -0.20261438   0.071895425   0.026143791    0.11111111
    0.47712418    0.37908497   -0.27124183   -0.27777778
   -0.18954248    -0.2875817    0.39542484   0.055555556
    0.16339869  0.0065359477   -0.13398693   0.055555556


d =d*d.inverse();
d=
             1             0             0             0
             0             1             0             0
             0             0             1             0
             0             0             0             1



C:\file-fy\project\cpp\CPPexperiment\Debug\experiment3.exe (
进程 13896)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. . .
```

3、 矩阵的幂预算中，负次幂如何处理，请说明。

该矩阵记为 M，单位矩阵记为 E

1) 非负幂次：

由 $M^n = E \cdot M^n$，可调用 n 次矩阵操作符重载*即可；

2) 负幂次：

由 $M^n = \left(M^{-1}\right)^{-n} = E \cdot Inv(M)^{-n}$，可先求矩阵的逆，再调用-n 次操作符*即可。

三 实验总结与建议

实验实施过程：先构思；画框架图；编写伪代码；用实际代码实现；运行调试；debug 直至无错误并得到预期结果。

实验解决方案：编写各操作符重载函数以及矩阵的求逆等操作函数即可解决。

附录：源代码

```
#include<iostream>
#include <iomanip>
#include<string>
using namespace std;
class Matrix_4x4
{
private:
    double matrix[4][4];
public:
    //默认构造函数，初始化矩阵为单位阵
    Matrix_4x4()
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                matrix[i][j] = 0;
        for (int i = 0; i < 4; i++)matrix[i][i] = 1;
    };
    //带参数构造函数，用一个 4x4 的二维数组初始化
    Matrix_4x4(double a[4][4])
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                matrix[i][j] = a[i][j];
    };
    //拷贝构造函数
    Matrix_4x4(const Matrix_4x4& a)
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                matrix[i][j] = a.matrix[i][j];
    }
    //重载 = 运算符，参数为矩阵对象
    Matrix_4x4 operator=(const Matrix_4x4& a)
    {
        if (&a == this)return *this;
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                matrix[i][j] = a.matrix[i][j];
        return *this;
    }
    //重载 = 运算符，参数为一个 4x4 的二维数组
    Matrix_4x4 operator=(double a[4][4])
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                matrix[i][j] = a[i][j];
        return *this;
    }
    // 重载算术运算符 + - * ^
    friend Matrix_4x4 operator+(const Matrix_4x4& a, const Matrix_4x4& b);
    friend Matrix_4x4 operator-(const Matrix_4x4& a, const Matrix_4x4& b);
    friend Matrix_4x4 operator*(const Matrix_4x4& a, const Matrix_4x4& b);
    friend Matrix_4x4 operator^(const Matrix_4x4& a,int n);
    // 重载[ ]运算符，实现双下标方式访问矩阵元素（该功能已经实现，无需自己写）
    const double* operator[] (const int i) const { return matrix[i]; }
    double* operator[] (const int i) { return matrix[i]; }
    // 重载输入<< 和输出 >>
    friend ostream& operator <<(ostream& output, Matrix_4x4& a)
    {
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                output.fill(' ');
                output.width(14);
```

```
                output.precision(8);
                if(abs(a.matrix[i][j])<1e-8)
                    output << 0;
                else
                    output << a.matrix[i][j];
            }
            output << endl;
        }
        output << endl;
        return output;
    }
    friend istream& operator >>(istream& input, Matrix_4x4& a)
    {
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
                input >> a.matrix[i][j];
        }
        return input;
    }
    //矩阵转置，不改变当前矩阵值，返回转置矩阵
    Matrix_4x4 transpose();
    //矩阵求逆，不改变当前矩阵值，返回逆矩阵
    Matrix_4x4 inverse();
    //line2 行加到 line1 行 以及对角线元化为 1
    void add_multiply(Matrix_4x4& a, int line1 = -1, int line2 = -1);
    //行变换将 line，col 位置的元置为 0
    void to0(Matrix_4x4& a, int line, int col);
};
Matrix_4x4 operator+(const Matrix_4x4& a, const Matrix_4x4& b)
{
    Matrix_4x4 c;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            c.matrix[i][j] = a.matrix[i][j] + b.matrix[i][j];
    return c;
}
Matrix_4x4 operator-(const Matrix_4x4& a, const Matrix_4x4& b)
{
    Matrix_4x4 c;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            c.matrix[i][j] = a.matrix[i][j] - b.matrix[i][j];
    return c;
}
Matrix_4x4 operator*(const Matrix_4x4& a, const Matrix_4x4& b)
{
    Matrix_4x4 c;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
        {
            double sum = 0;
            for (int k = 0; k < 4; k++)
                sum += a.matrix[i][k] * b.matrix[k][j];
            c.matrix[i][j] =sum;
        }
    return c;
}
Matrix_4x4 operator^(Matrix_4x4& a, int n)
{
    if (n>=0)
    {
        Matrix_4x4 b;
        for (int i = 1; i <= n; i++)
            b = b * a;
        return b;
    }
    else
    {
```

```
        Matrix_4x4 b;
        for (int i = 1; i <= -n; i++)
            b = b * a.inverse();
        return b;
    }
}
Matrix_4x4 Matrix_4x4::transpose()
{
    Matrix_4x4 a;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            a.matrix[i][j] = matrix[j][i];
    return a;
}
Matrix_4x4 Matrix_4x4::inverse()
{
    Matrix_4x4 copy(*this);
    Matrix_4x4 eye;
    double temp[4] = { 0 };

    //实现下三角为0，对角线为0
    for (int i = 0; i < 4; i++)
    {
        if (matrix[i][i] == 0)//对角线元为0
        {
            for (int j = 0; j < 4; j++)
            {
                if (matrix[j][i] != 0)
                {
                    copy.add_multiply(eye, i, j);
                }
            }
        }
        copy.add_multiply(eye, i);//主对角线元化为1
        for (int j = i + 1; j < 4; j++)//主元以下化为0
            copy.to0(eye, j, i);
    }
    //实现上三角为0
    for (int i = 0; i < 4; i++)
    {
        for (int j = i - 1; j >= 0; j--)//主元以上化为0
            copy.to0(eye, j, i);
    }
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)
                if (copy.matrix[i][j] != 1)
                    exit(1);
            if (i != j)
                if (copy.matrix[i][j] != 0)
                    exit(1);
        }

    }
    return eye;
}
void Matrix_4x4::to0(Matrix_4x4& a, int line, int col)
{
    if (line == col)return;
    double temp = matrix[line][col] / matrix[col][col];
    for (int i = 0; i < 4; i++)
    {
        matrix[line][i] -= temp * matrix[col][i];
        a.matrix[line][i] -= temp * a.matrix[col][i];
    }
}
void Matrix_4x4::add_multiply(Matrix_4x4& a, int line1, int line2)
```

```
{
    if (line2 != -1)//line2 加到 line1
    {
        for (int i = 0; i < 4; i++)
        {
            matrix[line1][i] += matrix[line2][i];
            a.matrix[line1][i] += a.matrix[line2][i];
        }
    }
    else//将第 line1 行主元变成 1
    {
        double temp = matrix[line1][line1];
        for (int i = 0; i < 4; i++)
        {
            matrix[line1][i] /= temp;
            a.matrix[line1][i] /= temp;
        }
    }
}
int main()
{
    double m[4][4] = { {1,2,3,4}, {8,6, 7,9}, {4,10,-4,12}, {-13,14, 45,28} };
    double  n[4][4] = { { 1,2,3,5 },{ 6,3,2,1},{ 4,3,5,2},{6,1,3,8} };
    //测试构造函数
    Matrix_4x4 a;
    Matrix_4x4 b(a);
    Matrix_4x4 c(m);
    Matrix_4x4 d;
    Matrix_4x4 e(n);
    Matrix_4x4 input_test;
    //测试下标重载
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            d[i][j] = m[i][j];
    //测试输出
    cout << "a= " << endl << a << endl;
    cout << "b(a)= " << endl << b << endl;
    cout << "c(m)= " << endl << c << endl;
    cout << "d= " << endl << d << endl;
    cout << "e(n)= " << endl << e << endl;
    //测试输入
    cout << "please input a matrix(4*4) to assign the object input_test:" << endl;
    cin >> input_test;
    cout << "input_test= " << endl << input_test << endl;
    //测试运算符重载
    d = a;
    cout << "d=a;" << endl << "d= " << endl << d << endl;
    d = m;
    cout << "d=m;" << endl << "d= " << endl << d << endl;
    d = a + c;
    cout << "d=a+c;" << endl << "d= " << endl << d << endl;
    d = a - b;
    cout << "d=a-b;" << endl << "d= " << endl << d << endl;
    d = c * e;
    cout << "d=c*e;" << endl << "d= " << endl << d << endl;
    for (int i = -3; i <= 3; i++)
    {
        d = e ^ i;
        cout << "d =e^" << i << ";" << endl << "d= " << endl << d << endl;
    }
    //测试转置、求逆等运算
    d = e.transpose();
    cout << "d=e.transpose();" << endl << "d= " << endl << d << endl;
    d = e.inverse();
    cout << "d=e.inverse();" << endl << "d= " << endl << d << endl;
    d = d * d.inverse();
    cout << "d =d*d.inverse();" << endl << "d= " << endl << d << endl;
    return 0;
}
```