Experiment4-董皓彧

环境:

```
g++.exe (x86_64-win32-seh-rev1, Built by MinGW-Builds project) 13.2.0
Visual Stdio Code 1.86.2
```

作业仓库地址:
https://github.com/FHYQ-Dong/Tsinghua-Program-Design-Assignments-2/tree/main/Experiment4

必做题

Experiment4-1

题目:

```
分析复制构造函数与类型转换构造函数的区别与联系？总结一下到目前为止学习了哪些类型的构造函数？各有什么特点？
```

答:

```
联系：二者均是构造函数，均可以被隐式调用 区别：复制构造函数由本类的另一个对象构造当前对象，而转换构造函数由其他类型的数据构造当前对象。复制构造函数的参数必须为对象的引用，而转换构造函数可以不是。
目前学习了默认构造函数、构造函数、复制构造函数、转换构造函数。
默认构造函数：由系统默认生成，可能存在“浅拷贝”问题。
构造函数：自己写。
复制构造函数：由本类的其他对象创建新对象时自动调用。
转换构造函数：谨慎使用，可能出现并不理想的类型转换问题。
```

Experiment4-2

题目:

```
编写一个程序,用成员函数重载运算符 “+” 和 “-” 将两个二维数组相加和相减，要求第一个二维数据的值由构造函数来设定，第二个二维数据的值由键盘输入。
```

输入格式:

```
一个 3x3 的二维数组，每个元素之间用空格隔开，一共 3 行
```

输出格式:

```
相加减的结果
```

代码:

```
#include <iostream>
```

```
#include <cstring>

template <typename T>
class Matrix {
    private:
        size_t rows, cols;
        T* data;

    template<typename U>
    friend std::istream& operator >> (std::istream& os, const Matrix<U>& m);
    template<typename U>
    friend std::ostream& operator << (std::ostream& os, const Matrix<U>& m);

    public:
        Matrix(size_t rows=1, size_t cols=1) : rows(rows), cols(cols) {
            data = new T[rows*cols];
            memset(data, 0, rows*cols*sizeof(T));
        }
        Matrix(const Matrix<T>& m) : rows(m.rows), cols(m.cols) {
            data = new T[rows*cols];
            memcpy(data, m.data, rows*cols*sizeof(T));
        }
        Matrix(T* d, size_t rows, size_t cols) : rows(rows), cols(cols) {
            data = new T[rows*cols];
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    data[i*cols+j] = d[i*cols+j];
                }
            }
        }
        ~Matrix() {
            delete[] data;
        }
        Matrix<T> operator + (const Matrix<T>& m) const {
            if (rows != m.rows || cols != m.cols) throw("Invalid matrix
addition");
            Matrix<T> result(rows, cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    result.data[i*cols+j] = data[i*cols+j] + m.data[i*cols+j];
                }
            }
            return result;
        }
        Matrix<T> operator - (const Matrix<T>& m) const {
            if (rows != m.rows || cols != m.cols) throw("Invalid matrix
subtraction");
            Matrix<T> result(rows, cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    result.data[i*cols+j] = data[i*cols+j] - m.data[i*cols+j];
                }
            }
            return result;
        }
        Matrix<T> operator * (const T& s) const {
            Matrix<T> result(rows, cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    result.data[i*cols+j] = data[i*cols+j] * s;
                }
            }
            return result;
        }
        Matrix<T> operator / (const T& s) const {
            Matrix<T> result(rows, cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    result.data[i*cols+j] = data[i*cols+j] / s;
                }
            }
            return result;
        }
        Matrix<T> operator * (const Matrix<T>& m) const {
            if (cols != m.rows) throw("Invalid matrix multiplication");
            Matrix<T> result(rows, m.cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < m.cols; j++) {
                    for (size_t k = 0; k < cols; k++) {
                        result.data[i*m.cols+j] += data[i*cols+k] *
m.data[k*m.cols+j];
                    }
                }
            }
            return result;
        }
        Matrix<T> operator += (const Matrix<T>& m) {
            if (rows != m.rows || cols != m.cols) throw("Invalid matrix
addition");
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    data[i*cols+j] += m.data[i*cols+j];
                }
            }
            return *this;
        }
        Matrix<T> operator -= (const Matrix<T>& m) {
            if (rows != m.rows || cols != m.cols) throw("Invalid matrix
subtraction");
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    data[i*cols+j] -= m.data[i*cols+j];
                }
            }
            return *this;
        }
        Matrix<T> operator *= (const T& s) {
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    data[i*cols+j] *= s;
                }
            }
            return *this;
        }
        Matrix<T> operator /= (const T& s) {
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    data[i*cols+j] /= s;
                }
            }
            return *this;
        }
        Matrix<T> operator *= (const Matrix<T>& m) {
            if (cols != m.rows) throw("Invalid matrix multiplication");
            Matrix<T> result(rows, m.cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < m.cols; j++) {
                    for (size_t k = 0; k < cols; k++) {
                        result.data[i*m.cols+j] += data[i*cols+k] *
m.data[k*m.cols+j];
                    }
                }
            }
            return result;
        }
        T* operator [] (size_t i) {
            return &data[i*cols];
        }
        Matrix<T> trans() const {
            Matrix<T> result(cols, rows);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    result.data[j*rows+i] = data[i*cols+j];
                }
            }
            return result;
        }
        Matrix<T> cofactor(size_t r, size_t c) const {
            if (rows == 1 || cols == 1) throw("Invalid matrix cofactor");
            Matrix<T> result(rows-1, cols-1);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    if (i == r || j == c) continue;
                    result.data[(i<r?i:i-1)*result.cols+(j<c?j:j-1)] =
data[i*cols+j];
                }
            }
            return result;
        }
        Matrix<T> det() const {
            if (rows != cols) throw("Invalid matrix determinant");
            if (rows == 1) return data[0];
            if (rows == 2) return data[0]*data[3] - data[1]*data[2];
            T result = 0;
            for (size_t i = 0; i < rows; i++) {
                result += data[i*cols] * cofactor(i, 0);
            }
            return result;
        }
        Matrix<T> inv() const {
            if (rows != cols) throw("Invalid matrix inverse");
            T d = det();
            if (d == 0) throw("Invalid matrix inverse");
            Matrix<T> result(rows, cols);
            for (size_t i = 0; i < rows; i++) {
                for (size_t j = 0; j < cols; j++) {
                    result.data[i*cols+j] = cofactor(i, j) / d;
                }
            }
            return result;
        }
};

template<typename U>
std::istream& operator >> (std::istream& os, const Matrix<U>& m) {
    for (size_t i = 0; i < m.rows; i++) {
        for (size_t j = 0; j < m.cols; j++)
            os >> m.data[i*m.cols+j];
    }
    return os;
}
template<typename U>
std::ostream& operator << (std::ostream& os, const Matrix<U>& m) {
    for (size_t i = 0; i < m.rows; i++) {
        for (size_t j = 0; j < m.cols; j++)
            os << m.data[i*m.cols+j] << ' ';
        os << std::endl;
    }
    return os;
}


int main() {
    int a[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    Matrix<int> m1((int*)(a), 3, 3), m2(3, 3);
    std::cin >> m2;
    std::cout << "m1 + m2 =\n" << (m1 + m2) << std::endl;
    std::cout << "m1 - m2 =\n" << (m1 - m2) << std::endl;
    return 0;
}
```

输入1：

```
1 1 1
1 1 1
1 1 1
```

输出1：

```
m1 + m2 =
2 3 4
5 6 7
8 9 10

m1 - m2 =
0 1 2
3 4 5
6 7 8
```

选做题

Optional-Experiment4-1

题目：

```
编写一个三维向量类 vector3 ，重载运算符 “*”，实现两个向量间的外积，重载为友元函数
```

输入格式：

```
无
```

输出格式：

```
略
```

代码：

```
#include <iostream>
#include <cmath>

template <typename T>
class vector3 {
    private:
        T x, y, z;

    template <typename U>
    friend vector3<U> operator*(const vector3<U>& v1, const vector3<U>& v2);

    public:
        vector3(T x=(T)(0), T y=(T)(0), T z=(T)(0)) : x(x), y(y), z(z) {}
        vector3(const vector3<T>& v) : x(v.x), y(v.y), z(v.z) {}
        ~vector3() {}
        void print() const {
            std::cout << "x: " << x << ", y: " << y << ", z: " << z <<
std::endl;
        }
        vector3<T> operator+(const vector3<T>& v) const {
            return vector3<T>(x+v.x, y+v.y, z+v.z);
        }
        vector3<T> operator-(const vector3<T>& v) const {
            return vector3<T>(x-v.x, y-v.y, z-v.z);
        }
        vector3<T> operator*(const T& s) const {
            return vector3<T>(x*s, y*s, z*s);
        }
        vector3<T> operator/(const T& s) const {
            return vector3<T>(x/s, y/s, z/s);
        }
        T dot(const vector3<T>& v) const {
            return x*v.x + y*v.y + z*v.z;
        }
        vector3<T> cross(const vector3<T>& v) const {
            return vector3<T>(y*v.z - z*v.y, z*v.x - x*v.z, x*v.y - y*v.x);
        }
        T length() {
            return sqrt(x*x + y*y + z*z);
        }
        vector3<T> normalize() const {
            T l = length();
            return vector3<T>(x/l, y/l, z/l);
        }
};

template <typename U>
vector3<U> operator*(const vector3<U>& v1, const vector3<U>& v2) {
    return vector3<U>(v1.y*v2.z - v1.z*v2.y, v1.z*v2.x - v1.x*v2.z, v1.x*v2.y -
v1.y*v2.x);
}

inline void test() {
    vector3<int> v1(1, 2, 3);
    vector3<int> v2(4, 5, 6);
    vector3<int> v3 = v1 * v2;
    v3.print();
}

int main() {
    test();
    return 0;
}
```

输入1：

```

```

输出1：

```
x: -3, y: 6, z: -3
```

Optional-Experiment4-2

题目：

```
编写 Date 类，包含年、月、日，并重载 “++” 运算符（前置后置都需要重载），实现日期增加 1 天，重载为成员函数
```

输入格式：

```
无
```

输出格式：

```
略
```

代码：

```cpp
#include <iostream>
#include <ctime>

class Date {
    private:
        int year, month, day;
        bool is_leap;
        static constexpr int daysInMonth[][2] = {
            {31, 31}, {28, 29}, {31, 31}, {30, 30}, {31, 31}, {30, 30},
            {31, 31}, {31, 31}, {30, 30}, {31, 31}, {30, 30}, {31, 31}
        };

        bool is_leap_year() const {
            return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
        }

        void normalize() {
            is_leap = is_leap_year();
            while (day > daysInMonth[month-1][is_leap]) {
                day -= daysInMonth[month-1][is_leap];
                month++;
                while (month > 12) {
                    month -= 12;
                    year++;
                    is_leap = is_leap_year();
                }
            }
            while (day < 1) {
                month--;
                while (month < 1) {
                    month += 12;
                    year--;
                    is_leap = is_leap_year();
                }
                day += daysInMonth[month-1][is_leap];
            }
        }

    public:
        Date(int year=1970, int month=1, int day=1) : year(year), month(month),
day(day) {
            normalize();
        }
        Date(const Date& d) : year(d.year), month(d.month), day(d.day) {}
        Date(const time_t& t) {
            struct tm* timeinfo = localtime(&t);
            day = timeinfo->tm_mday;
            month = timeinfo->tm_mon + 1;
            year = timeinfo->tm_year + 1900;
            normalize();
        }
        ~Date() {}
        void print(const char* sep) const {
            std::cout << year << sep << month << sep << day << std::endl;
        }
        Date& operator = (const Date& d) {
            day = d.day;
            month = d.month;
            year = d.year;
            return *this;
        }
        Date operator + (const Date& d) const {
            Date result;
            result.day = day + d.day;
            result.month = month + d.month;
            result.year = year + d.year;
            result.normalize();
            return result;
        }
        Date operator - (const Date& d) const {
            Date result;
            result.day = day - d.day;
            result.month = month - d.month;
            result.year = year - d.year;
            result.normalize();
            return result;
        }
        bool operator == (const Date& d) const {
            return day == d.day && month == d.month && year == d.year;
        }
        bool operator != (const Date& d) const {
            return day != d.day || month != d.month || year != d.year;
        }
        bool operator < (const Date& d) const {
            if (year < d.year) return true;
            if (year > d.year) return false;
            if (month < d.month) return true;
            if (month > d.month) return false;
            return day < d.day;
        }
        bool operator > (const Date& d) const {
            if (year > d.year) return true;
            if (year < d.year) return false;
            if (month > d.month) return true;
            if (month < d.month) return false;
            return day > d.day;
        }
        bool operator <= (const Date& d) const {
            return *this < d || *this == d;
        }
        bool operator >= (const Date& d) const {
            return *this > d || *this == d;
        }
        Date& operator += (const Date& d) {
            day += d.day;
            month += d.month;
            year += d.year;
            normalize();
            return *this;
        }
        Date& operator -= (const Date& d) {
            day -= d.day;
            month -= d.month;
            year -= d.year;
            normalize();
            return *this;
        }
        Date& operator ++ () {
            day++;
            normalize();
            return *this;
        }
        Date operator ++ (int) {
            Date temp(*this);
            operator++();
            return temp;
        }
};

inline void test() {
    Date d1(2024, 2, 28);
    d1.print("/");
    d1++;
    d1.print("/");
    ++d1;
    d1.print("/");
}

int main() {
    test();
    return 0;
}
```

输入1：

```

```

输出1：

```
2024/2/28
2024/2/29
2024/3/1
```