模板编程 上机练习

李宇豪 21305412

任务：

矩阵或向量可以是双精度型（double）、单精度型（float），甚至整数型（int）。对大规模的矩阵选择合适的数据类型可以更合理地使用内存。前面的练习中矩阵和向量被默认定义为双精度型，希望在保持代码基本不变的情况下，可根据需要定义双精度、单精度、整数型的矩阵。

如：

Matrix<double> m(3,3); //双精度型的矩阵
Matrix<float> m(3,3); //单精度型的矩阵
Matrix<int> m(3,3); //双精度型的矩阵

要求声明和实现分离。

思路：

采用类模板定义 Matrix 类。

要点：

（1） **声明**：在 Matrix.h 头文件中声明类模板及其成员

（2） **实现**：在 Matrix.cpp 文件中实现类模板的成员函数
其中友元函数在 Matrix.h 头文件中采用内联函数的方式实现

（3） **显式实例化**：类模板的声明和实现分开，在.cpp 文件最后添加实例化声明

源代码：

Matrix.h 文件中的代码

```
#include <iostream>
using namespace std;

//不包含相应的头文件，仅声明会使用，避免重复包含头文件的简单方法
class Vector;


//类模板声明
template<typename T>
class Matrix
{
public:
    //构造函数、拷贝构造函数与析构函数
   Matrix():row(1),col(1) {}
   Matrix(int _row, int _col);
   Matrix(const Matrix& _ma);
   ~Matrix()
   {
      delete[] p_data;
   }
```

```
 //读取成员数据的相关函数
int getrow() const { return row; }
int getcol() const { return col; }
 T getdata(int i, int j) const;

 //所有元素统一初始化为某个值
 void init(T value);
 //返回对矩阵的某个元素的引用，方便对其赋值
 T& data(int i, int j);

 //算符重载
 //直接给矩阵赋值的 = 算符重载
Matrix<T> operator=(T d);
 Matrix<T> operator=(Matrix<T> m);

//矩阵加法 + 算符重载
 Matrix<T> operator+(const Matrix<T> m);
 //矩阵与实数相加  类模板中的友元函数采用内联的方式实现
friend Matrix<T> operator+(const Matrix<T> m, T d)
 {
    Matrix<T> m2(m);
    for (int i = 0; i < m2.row ; i++)
        for(int j=0;j< m2.col;j++)
            m2.data(i,j) += d;
    return m2;
 }
 //实数与矩阵相加  类模板中的友元函数采用内联的方式实现
friend Matrix<T> operator+(T d, const Matrix<T> m)
 {
    Matrix<T> m2(m);
    for (int i = 0; i < m2.row; i++)
        for (int j = 0; j< m2.col; j++)
            m2.data(i, j) += d;
    return m2;
 }

 //矩阵乘法 * 算符重载
 Matrix<T> operator*(const Matrix<T> m);
 //矩阵与实数相乘  类模板中的友元函数采用内联的方式实现
 friend Matrix<T> operator*(const Matrix<T> m, T d)
 {
    Matrix<T> m2(m);
    for (int i = 0; i < m2.row; i++)
        for (int j = 0; j < m2.col; j++)
            m2.data(i, j) = d * m.getdata(i, j);
    return m2;
 }
```

```
    //实数与矩阵相乘  类模板中的友元函数采用内联的方式实现
    friend Matrix<T> operator*(T d, const Matrix<T> m)
     {
         Matrix<T> m2(m);
         for (int i = 0; i < m2.row; i++)
             for (int j = 0; j < m2.col; j++)
                 m2.data(i, j) = d * m.getdata(i, j);
         return m2;
     }

    //old fashion 矩阵乘法函数
     Matrix<T> multiply(Matrix<T>& _ma);
    Vector& multiply(Vector& _vec);
     Matrix<T>& multiply2(Matrix<T>& _ma);

    //old fashion 矩阵打印函数
    void print();

protected:
    int    row, col;
    T* p_data;
};
```

Matrix.cpp 文件中的代码

```
#include "matrix.h"
#include "vector.h"

//构造函数
template<typename T>
Matrix<T>::Matrix(int _row, int _col) {
    row = _row;
    col = _col;
    p_data = new T[row * col];
}

//拷贝构造函数
template<typename T>
Matrix<T>::Matrix(const Matrix& _ma)
{
    row = _ma.row;
    col = _ma.col;
    p_data = new T[row * col];
    for (int i = 0; i < row * col; i++)
      p_data[i] = _ma.p_data[i];
}
```

```cpp
//给矩阵赋值的 = 算符重载
template<typename T>
Matrix<T> Matrix<T>::operator=(T d)
{
   for (int i = 0; i < row * col; i++)
      p_data[i] = d;
   return *this;
}
//给矩阵赋值的 = 算符重载
template<typename T>
Matrix<T> Matrix<T>::operator=(Matrix<T> m)
{
   int r = m.getrow();
   int c = m.getcol();
   if (r != this->row || c != this->col)
   {
      cout << "Matrix does NOT match!" << endl;
      return *this;
   }
   for (int i = 0; i <r ; i++)
      for(int j=0;j<c;j++)
          this->data(i,j) = m.getdata(i,j);

   return *this;
}

//矩阵元素的初始化函数
template<typename T>
void Matrix<T>::init(T value)
{
   for (int i = 0; i < row * col; i++)
      p_data[i] = value;
}

//返回对矩阵某个元素的引用
template<typename T>
T Matrix<T>::getdata(int i, int j) const
{
   return p_data[i * col + j];
}

//读取矩阵元素
template<typename T>
T& Matrix<T>::data(int i, int j)
{
   return p_data[i * col + j];
}
```

```cpp
//矩阵加法 + 算符重载
template<typename T>
Matrix<T> Matrix<T>::operator+(const Matrix<T> m)
{
   Matrix<T> m2(m);
   int r = m2.getrow();
   int c = m2.getcol();
   if (r != this->row || c != this->col)
   {
      cout << "Matrix does NOT match!" << endl;
      return m2;//You can choose other exit method
   }
   for (int i = 0; i < m2.row; i++)
      for (int j = 0; j < m2.col; j++)
         m2.data(i, j) += this->getdata(i, j);

   return m2;
}

//矩阵乘法 * 算符重载
template<typename T>
Matrix<T> Matrix<T>::operator*(const Matrix<T> m)
{
   if (col != m.row)
      cerr << "this matrix.col != _ma.row" << endl;

   int rownew = row, colnew = m.col;
   Matrix<T> c(rownew, colnew);
    for(int i = 0; i < row ; i++)
    {
        for(int j = 0; j < m.col ; j++)
        {
            T sum = 0;
            for(int k = 0; k < col; k++)
            {
                sum += p_data[i * col + k] * m.p_data[k * m.col + j];
            }
            c.data(i,j) = sum;
        }
    }
   return c;
}


//矩阵乘法函数的实现
template<typename T>
Matrix<T> Matrix<T>::multiply(Matrix<T>& _ma)
```

```
{
   if (col != _ma.row)
      cerr << "this matrix.col != _ma.row" << endl;

   int rownew = row, colnew = _ma.col;
   Matrix<T> c(rownew, colnew);
   for (int i = 0; i < rownew; i++)
   {
      for (int j = 0; j < colnew; j++)
      {
         T sum = 0.;
         for (int k = 0; k < col; k++)
            sum += p_data[i * col + k] * _ma.p_data[k * _ma.col + j];

         c.p_data[i * colnew + j] = sum;
      }
   }
   return c;
}

//矩阵乘法函数的实现 矩阵×向量
template<typename T>
Vector& Matrix<T>::multiply(Vector& _vec)
{
   if (col != _vec.getsize())
      cerr << "size is not correct!" << endl;
   Vector* c = new Vector(row);
   for (int i = 0; i < row; i++)
   {
      T sum = 0.;
      for (int j = 0; j < col; j++)
         sum += p_data[i * col + j] * _vec.data(j);
      c->data(i) = sum;
   }
   return *c;
}

//矩阵乘法函数的实现 矩阵×矩阵 返回对矩阵的引用
template<typename T>
Matrix<T>& Matrix<T>::multiply2(Matrix<T>& _ma)
{
   if (col != _ma.row)
      cerr << "this matrix.col != _ma.row" << endl;

   int rownew = row, colnew = _ma.col;
   Matrix<T>* c = new Matrix(rownew, colnew);
   for (int i = 0; i < rownew; i++)
```

```
    {
        for (int j = 0; j < colnew; j++)
        {
            T sum = 0.;
            for (int k = 0; k < col; k++)
                sum += p_data[i * col + k] * _ma.p_data[k * _ma.col + j];

            c->p_data[i * colnew + j] = sum;
        }
    }
    return *c;
}

//矩阵打印函数
template<typename T>
void Matrix<T>::print()
{
    cout << "Output matrix " << endl;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            cout << p_data[i * col + j] << "  ";
        cout << endl;
    }
}

//将类模板显式实例化 int、float、double 三种类型
template class Matrix<double>;
template class Matrix<float>;
template class Matrix<int>;
```

Main.cpp 文件中的代码

```
#include <iostream>
#include "matrix.h"
using namespace std;

int    main()
{
    int n = 3;
    double A[3][3] = { {1.,1., 1.},{1.,3.,-2.},{2.,-2.,1.} },
           b[] = { 6., 1., 1. };

    //初始化单精度型矩阵
    Matrix<float>  m(n,n);
```

```
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            m.data(i, j) = A[i][j];
    cout << "初始化单精度型矩阵" << endl;
    m.print();

    cout << "测试矩阵加法运算符重载 m2=m+1.5" << endl;
    Matrix<float>  m2=m+1.5;
    m2.print();

    cout << "测试矩阵乘法运算符重载 m3=1.5*m" << endl;
    Matrix<float>  m3=1.5*m;
    m3.print();

    return 0;
}
```

代码运行结果：