代码：

#include<iostream>

#include<cmath>

#include<cstdlib>

using namespace std;

class Complex

{

double R;

double I;

public:

double GetR() const { return R; };

void SetR(double r) { R = r; }

double GetI() const { return I; };

void SetI(double i) { I = i; }

Complex() { R = I = 0; }//构造函数

Complex(double r) { R = r; I = 0; }//如果既有重载又有默认值，调用单参时会报错

Complex(double r, double i) {

R = r;

I = i;

}

~Complex() {}//析构函数，在对象被销毁时调用，释放内存

double GetModelSquare() const {

return R \* R + I \* I;

}

double GetModel() const {

return sqrt(GetModelSquare());

}

bool operator == (Complex c1) const {

return this->R == c1.R && this->I == c1.I;

}

Complex& operator = (const Complex& another)

{

this->R = another.R;

this->I = another.I;

return \*this;

}

double operator[](int i) const

{

if (i == 0) return R;

if (i == 1) return I;

return 0;

}

double& operator[](int i)

{

if (i == 0) return R;

if (i == 1) return I;

}

Complex& operator++()

{

++R;

return \*this;

}

Complex operator++(int)

{

Complex c = \*this;

++R;

return c;

}

Complex& operator--()

{

--R;

return \*this;

}

Complex operator--(int)

{

Complex c = \*this;

--R;

return c;

}

Complex& operator+=(const Complex& c1) {

this->R += c1.R;

this->I += c1.I;

return \*this;

}

Complex& operator-=(const Complex& c1) {

this->R -= c1.R;

this->I -= c1.I;

return \*this;

}

Complex& operator \*=(const Complex& c1) {

double r1 = this->R, i1 = this->I;

this->R = c1.R \* r1 - c1.I \* i1;

this->I = c1.I \* r1 + c1.R \* i1;

return \*this;

}

Complex& operator/=(const Complex& c1) {

double M = c1.GetModel();

double r1 = c1.GetR() \* this->R + c1.GetI() \* this->I;

double i1 = this->I \* c1.GetR() - this->R \* c1.GetI();

this->I = i1 / M;

this->R = r1 / M;

return \*this;

}

friend bool operator != (const Complex& c1, const Complex& c2);

/\*friend Complex operator \*(const Complex& c1, const Complex& c2)//友元，使全局函数可以直接访问私有数据 \*/

};

const Complex i(0, 1);

bool operator != (const Complex& c1, const Complex& c2)

{

return c1.R != c2.R || c1.I != c2.I;

}

Complex operator \*(const Complex& c1, const Complex& c2) {//定义运算符函数

double R = c1.GetR() \* c2.GetR() - c1.GetI() \* c2.GetI();

double I = c1.GetI() \* c2.GetR() + c1.GetR() \* c2.GetI();

return Complex(R, I);

}

Complex operator +(const Complex& c1, const Complex& c2) {

double R = c1.GetR() + c2.GetR();

double I = c1.GetI() + c2.GetI();

return Complex(R, I);

}

Complex operator -(const Complex& c1, const Complex& c2) {

double R = c1.GetR() - c2.GetR();

double I = c1.GetI() - c2.GetI();

return Complex(R, I);

}

Complex operator /(const Complex& c1, const Complex& c2) {

double M = c1.GetModel();

double R = c1.GetR() \* c2.GetR() + c1.GetI() \* c2.GetI();

double I = c1.GetI() \* c2.GetR() - c1.GetR() \* c2.GetI();

return Complex(R / M, I / M);

}

Complex operator ^(const Complex& c1, int a) {

double R = c1.GetR();

double I = c1.GetI();

for (int j = a; j > 1; j--) {

double r = R, i = I;

R = r \* c1.GetR() - i \* c1.GetI();

I = i \* c1.GetR() + r \* c1.GetI();

}

return Complex(R, I);

}

ostream& operator <<(ostream& o, const Complex& c) {

if (c.GetR() == 0 && c.GetI() == 0)o << "0";

else if (c.GetR() == 0)o << c.GetI() << "i";

else if (c.GetI() == 0)o << c.GetR();

else o << c.GetR()

<< (c.GetI() > 0 ? " + " : " - ")

<< abs(c.GetI()) << "i";

return o;

}

template<typename T>

class Matrix {

private:

int row,col;

int len;

T \* Mat;

public:

Matrix<T>() { row = 1; col = 1; len = 1; Mat = new T[len]; }

Matrix<T>(const int r, const int c, const T\* M, const int l) {

row = r; col = c;

len = l;

delete[] Mat;

Mat = new T[len];

for (int i = 0; i < len; i++) {

Mat[i] = M[i];

}

}

Matrix<T>(const Matrix<T>& m1) {

row = m1.row;

col = m1.col;

len = m1.len;

Mat = new T[len];

for (int i = 0; i < len; i++) {

Mat[i] = m1.Mat[i];

}

}

~Matrix() {

delete[] Mat;

//cout << "deleted~" << endl;

};

Matrix<T>& operator =(const Matrix<T>& m1) {

row = m1.row;

col = m1.col;

len = m1.len;

delete []Mat;

//cout << "deleted=" << endl;

Mat = new T[len];

for (int i = 0; i < len; i++) {

Mat[i] = m1.Mat[i];

}

return \*this;

}

template<typename t>

friend Matrix<t> operator+(const Matrix<t>& m1, const Matrix<t>& m2);

template<typename t>

friend Matrix<t> operator-(const Matrix<t>& m1, const Matrix<t>& m2);

template<typename t>

friend Matrix<t> operator\*(const Matrix<t>& m1, const Matrix<t>& m2);

template<typename t>

friend Matrix<t>& operator+=(Matrix<t>& m1, const Matrix<t>& m2);

template<typename t>

friend Matrix<t>& operator-=(Matrix<t>& m1, const Matrix<t>& m2);

template<typename t>

friend Matrix<t>& operator\*=(Matrix<t>& m1, const Matrix<t>& m2);

template<typename t>

friend Matrix<t> operator~(const Matrix<t>& m1);

template<typename t>

friend ostream& operator <<(ostream& o, Matrix<t> M);

};

template<typename T>

Matrix<T> operator+(const Matrix<T>& m1,const Matrix<T>& m2) {

Matrix<T> M(m1.row,m1.col,m1.Mat,m1.len);

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m1.col; j++) {

M.Mat[i \* m1.col + j]=m1.Mat[i \* m1.col + j]+ m2.Mat[i \* m1.col + j];

}

}

return M;

}

template<typename T>

Matrix<T> operator-(const Matrix<T>& m1,const Matrix<T>& m2) {

Matrix<T> M(m1.row, m1.col, m1.Mat, m1.len);

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m1.col; j++) {

M.Mat[i \* m1.col + j] = m1.Mat[i \* m1.col + j] - m2.Mat[i \* m2.col + j];

}

}

return M;

}

template<typename T>

Matrix<T> operator\*(const Matrix<T>& m1,const Matrix<T>& m2) {

Matrix<T> M(m1.row, m2.col, m1.Mat, m1.row\*m2.col);

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m2.col; j++) {

for (int k = 0; k < m1.col; k++) {

M.Mat[i \* m2.col + j] += m1.Mat[i \* m1.col + k] \* m2.Mat[k \* m1.col + j];

}

}

}

return M;

}

template<typename T>

Matrix<T>& operator+=(Matrix<T>& m1, const Matrix<T>& m2) {

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m1.col; j++) {

m1.Mat[i \* m1.col + j] += m2.Mat[i \* m2.col + j];

}

}

return m1;

}

template<typename T>

Matrix<T>& operator-=(Matrix<T>& m1, const Matrix<T>& m2) {

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m1.col; j++) {

m1.Mat[i \* m1.col + j] -= m2.Mat[i \* m2.col + j];

}

}

return m1;

}

template<typename T>

Matrix<T>& operator\*=(Matrix<T>& m1, const Matrix<T>& m2) {

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m2.col; j++) {

T temp = 0;

for (int k = 0; k < m1.col; k++) {

temp= m1.Mat[i \* m1.col + k] \* m2.Mat[k \* m1.col + j];

}

m1.Mat[i \* m2.col + j] = temp;

}

}

m1.len = m1.row \* m2.col;

m1.col = m2.col;

return m1;

}

template<typename T>

Matrix<T> operator~(const Matrix<T>& m1) {

Matrix<T> M(m1.col,m1.row,m1.Mat ,m1.len);

for (int i = 0; i < m1.row; i++) {

for (int j = 0; j < m1.col; j++) {

M.Mat[i \* m1.col + j] = m1.Mat[j \* m1.col + i];

}

}

return M;

}

template<typename T>

ostream& operator <<(ostream& o, const Matrix<T>& M) {

for (int i = 0; i < M.row; i++) {

o << "( ";

for (int j = 0; j < M.col; j++) {

o << M.Mat[i \* M.col + j];

if(j!=M.col-1)o<< " ";

}

o << ")"<<endl;

}

return o;

}

int main()

{

{

cout << "float - Matrix" << endl;

float a[9] = { 1, 3, 0, 1, 2, 0, -1, -1, -4 };

float b[9] = { 2, 0, 0, 0, -4, 0, 0, 0, 7 };

Matrix<float> m1(3, 3, a, 9);

cout << " m1=\n" << m1 << endl;

Matrix<float> m2(3, 3, b, 9);

cout << " m2=\n" << m2 << endl;

Matrix<float> m3 = m1 + m2;

cout << " m3=\n" << m3 << endl;

Matrix<float> m4 = m1 \* m2;

cout << " m4=\n" << m4 << endl;

Matrix<float> m5;

m5 = m4;

m4 -= m3;

cout << " m5=\n" << m5 << endl;

cout << " ~m5=\n" << ~m5 << endl; /\*转置\*/

}

{

cout << "Complex - Matrix" << endl;

Complex a[9] = { 1 + i , 3.1 - i, 0, 1 + 1.4 \* i, 2 + 3 \* i, 0, -1 + i, -1 - i, -4 - 0.5 \* i };

Complex b[9] = { 2.1 + i, 0, 0, 0, -4 - 3 \* i, 0, 0, 0, 7 - 0.3 \* i };

Matrix<Complex> m1(3, 3, a, 9);

//cout << " m1=\n" << m1 << endl;

Matrix<Complex> m2(3, 3, b, 9);

//cout << " m2=\n" << m2 << endl;

Matrix<Complex> m3 = m1 + m2;

cout << " m3=\n" << m3 << endl;

Matrix<Complex> m4 = m1 \* m2;

cout << " m4=\n" << m4 << endl;

Matrix<Complex> m5;

m5 = m4;

m4 -= m3;

cout << " m5=\n" << m5 << endl;

cout << " ~m5=\n" << ~m5 << endl; /\*转置\*/

}

return 0;

}

结果：

