第八章代码 运算符重载

/\*

\* ex8\_1.cpp

\*

\* 1．为第6 章习题中定义的Date 类、Employee 类和Student 类提供I/O 流输出操作符。

\*/

#include<string>

#include<vector>

#include<iostream>

using namespace std;

class Date{

int year = 1970;

int month = 1;

int day = 1;

public:

//constructor

Date(int y, int m, int d){

if (y > 0)

year = y;

else

year = 1970;

if(m >= 1 && m <= 12)

month = m;

else

month = 1;

if(leapYear() && month == 2){

if(d >= 1 && d <= 29)

day = d;

else

day = 1;

}

else{

if(d >= 1 && d <= days[month])

day = d;

else

day = 1;

}

}

//default constructor

Date() : Date(1970, 1, 1){}

friend ostream& operator<<(ostream& os, const Date& d);

private:

int days[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

bool leapYear() const{

return ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0));

}

};

ostream& operator<<(ostream& os, const Date& d){

os << d.year << "/" << d.month << "/" << d.day;

return os;

}

//----------------------------------------------------------

class Employee{

string name;

unsigned int id;

string department;

string title;

double salary = 0;

string address;

vector<string> phone;

public:

Employee(string ename, unsigned eid) : name(ename), id(eid){}

Employee(string ename, unsigned eid, string dept, string ttl)

: name(ename), id(eid), department(dept), title(ttl) {}

friend ostream& operator<<(ostream& os, const Employee& e);

};

ostream& operator<<(ostream& os, const Employee& e){

os <<"Name: " << e.name << "\n"

<<"ID: " << e.id << "\n"

<<"Department: " << e.department <<"\n"

<<"Title: "<< e.title << "\n";

cout << "Tel: ";

for(auto p : e.phone)

os << p << "\t";

return os;

}

//----------------------------------------------------------

class Student{

string name;

unsigned int id;

string department;

public:

Student(string sname, unsigned sid) : name(sname), id(sid){}

Student(string sname, unsigned sid, string dept) : name(sname), id(sid), department(dept){}

friend ostream& operator<<(ostream& os, const Student& s);

};

ostream& operator<<(ostream& os, const Student& s){

os << s.id <<"\t" << s.name <<"\t" << s.department;

return os;

}

//----------------------------------------------------------

int main()

{

Date d(2017, 2, 12);

cout << "Date\n" << d << endl;

Employee e("Lily", 123, "Sales", "Manager");

cout << "Employee\n" << e << endl;

Student s("LinDong", 118109, "Computer Science");

cout << "Student\n" << s << endl;

return 0;

}

/\*

\* ex8\_2.cpp

\*

\* 2．为第6 章习题中的复数类ComplexNumber 定义运算符函数，

\* 使得ComplexNumber的对象看起来可以像基本数据类型一样进行算术运算。

\*

\*

\*/

#include <iostream>

#include <sstream>

#include <string>

using namespace std;

class ComplexNumber{

public:

ComplexNumber(double r = 0, double i = 0): real(r), imag(i){}

double getImaginary() const;

double getReal() const;

ComplexNumber operator+(const ComplexNumber& c)const;

ComplexNumber operator-(const ComplexNumber& c)const;

ComplexNumber operator\*(const ComplexNumber& c)const;

ComplexNumber operator/(const ComplexNumber& c)const;

friend ostream& operator<<(ostream& os, const ComplexNumber& c);

friend istream& operator>>(istream& is, ComplexNumber& c);

private:

double real = 0;

double imag = 0;

string toString()const;

};

double ComplexNumber::getImaginary() const {

return imag;

}

double ComplexNumber::getReal() const {

return real;

}

string ComplexNumber::toString() const {

ostringstream os;

if(real == 0){

if(imag == 0)

os << 0;

else

os << imag << "i";

}

else{

if(imag == 0)

os << real;

else if(imag > 0)

os << real << "+" << imag << "i";

else

os << real << imag << "i";

}

return os.str();

}

ComplexNumber ComplexNumber::operator+(const ComplexNumber& c) const {

return ComplexNumber(real + c.real, imag + c.imag);

}

ComplexNumber ComplexNumber::operator-(const ComplexNumber& c) const {

return ComplexNumber(real - c.real, imag - c.imag);

}

//(a+bi) \* (c+di) = (ac-bd) + (bc + ad)i

//(a+bi) / (c+di) = (ac+bd)/(c2+d2) + (bc-ad)i/(c2+d2)

ComplexNumber ComplexNumber::operator\*(const ComplexNumber& c) const {

ComplexNumber cn;

cn.real = real \* c.real - imag \* c.imag;

cn.imag = imag \* c.real + real \* c.imag;

return cn;

}

ComplexNumber ComplexNumber::operator/(const ComplexNumber& c) const {

ComplexNumber cn;

double cds = c.real \* c.real + c.imag \* c.imag;

cn.real = (real \* c.real + imag \* c.imag) / cds;

cn.imag = (imag \* c.real - real \* c.imag) / cds;

return cn;

}

ostream& operator<<(ostream& os, const ComplexNumber& c){

os << c.toString();

return os;

}

istream& operator>>(istream& is, ComplexNumber& c){

is >> c.real >> c.imag;

return is;

}

//------------------------------

int main()

{

ComplexNumber c1(1, 2), c2(3, 4), c3(-3, 2), c4(2, -4), c5(-1, -2), c6;

cout <<"c1: " << c1 << endl;

cout <<"c2: " << c2 << endl;

cout <<"c3: " << c3 << endl;

cout <<"c4: " << c4 << endl;

cout <<"c5: " << c5 << endl;

cout <<"c6: " << c6 << endl;

cout <<"c1 + c2 = " << c1 + c2 << endl;

cout <<"c3 + c2 = " << c3 + c2 << endl;

cout <<"c4 + c2 = " << c4 + c2 << endl;

cout <<"c1 - c2 = " << c1 - c2 << endl;

cout <<"c1 \* c2 = " << c1 \* c2 << endl;

cout <<"c1 / c2 = " << c1 / c2 << endl;

cout << "Enter a complex number: (real imaginary): " ;

cin >> c6;

cout << c6 << endl;

return 0;

}

/\*

\* ex8\_3.cpp

\* ch8

\* 3．内置的算术类型不能表示所有的有理数，为此请定义一个有理数类Rational（提示：有理数可以表示为两个整数的比），重载需要的运算符，使之能够像内置类型一样使用。使

用示例如下：

\*/

#include <string>

#include <iostream>

#include <cassert>//assert

#include <sstream>

using namespace std;

//---------------------------------

const int SCALE = 100000;

class Rational {

public:

//constructor

Rational(int n, int d);

Rational(int n) :

Rational(n, 1) {

}

Rational() :

Rational(0, 1) {

}

Rational(double val);

//arithmatic operations算术操作

Rational operator-() const;

Rational operator+(const Rational& r) const;

Rational operator-(const Rational& r) const;

Rational operator\*(const Rational& r) const;

Rational operator/(const Rational& r) const;

//conversion转换 转化 换算

string toString() const;

operator double() const;

//overload 重载 operator << and >>

friend istream& operator>>(istream& in, Rational& r);

friend ostream& operator<<(ostream& out, const Rational& r);

//relational operations 关系操作

friend bool operator==(const Rational& left, const Rational& right);

friend bool operator!=(const Rational& left, const Rational& right);

friend bool operator<(const Rational& left, const Rational& right);

friend bool operator<=(const Rational& left, const Rational& right);

friend bool operator>(const Rational& left, const Rational& right);

friend bool operator>=(const Rational& left, const Rational& right);

private:

int deno = 1; //denominator（分母） cannot be zero

int nume = 0; //numerator（分子）

//private operation（私有操作）

static int gcd(int m, int n);

static int iabs(int n);

};

//---------------------------

Rational::Rational(int n, int d) { //validate

assert(d != 0);//宣称

if (n == 0) {

nume = 0;

deno = 1;

return;

}

int g = gcd(iabs(d), iabs(n));

if (d > 0) {

deno = d / g;

nume = n / g;

} else if (d < 0) {

deno = -1 \* d / g;

nume = -1 \* n / g;

} else {

deno = 1;

nume = 0;

}

}

Rational::Rational(double val) :

Rational(static\_cast<int>(val \* SCALE), SCALE) {

}

//arithmatic operations 算术操作

Rational Rational::operator-() const {

int d = deno;

int n = -1 \* nume;

return Rational(n, d);

}

Rational Rational::operator+(const Rational& r) const {

int d = deno \* r.deno;

int n = deno \* r.nume + nume \* r.deno;

return Rational(n, d);

}

Rational Rational::operator-(const Rational& r) const {

int d = deno \* r.deno;

int n = nume \* r.deno - deno \* r.nume;

return Rational(n, d);

}

Rational Rational::operator\*(const Rational& r) const {

int d = deno \* r.deno;

int n = nume \* r.nume;

return Rational(n, d);

}

Rational Rational::operator/(const Rational& r) const {

int d = deno \* r.nume;

int n = nume \* r.deno;

return Rational(n, d);

}

//conversion 转化

string Rational::toString() const {

ostringstream val;

val << nume;

if (deno != 1)

val << "/" << deno;

return val.str();

}

Rational::operator double() const {

return (nume \* 1.0) / deno;

}

int Rational::gcd(int m, int n) {

int g = m < n ? m : n;

while (g > 1) {

if (m % g == 0 && n % g == 0)

break;

--g;

}

return g;

}

int Rational::iabs(int n) {

if (n < 0)

return -1 \* n;

else

return n;

}

//relational operations

bool operator ==(const Rational& left, const Rational& right) {

Rational temp = left - right;

if(temp.nume == 0)

return true;

else

return false;

}

bool operator !=(const Rational& left, const Rational& right) {

return !(left == right);

}

bool operator <(const Rational& left, const Rational& right) {

Rational temp = left - right;

if(temp.nume < 0)

return true;

else

return false;

}

bool operator <=(const Rational& left, const Rational& right) {

return (left == right || left < right);

}

bool operator >(const Rational& left, const Rational& right) {

return (right < left)&&(left != right);

}

bool operator >=(const Rational& left, const Rational& right) {

return !(left < right);

}

//operator << and >>

istream& operator>>(istream& in, Rational& r) {

int n, d;

in >> n >> d;

r = Rational(n, d); //create a rational

return in;

}

ostream& operator<<(ostream& out, const Rational& r) {

out << r.toString();

return out;

}

int main() {

Rational r1; //r1=0

Rational r2(1, 2); //r2 = 1/2

Rational r3(-2, 3); //r3 = -2/3

Rational r4(1.5); //double 转换为Rational

cout << r1 << endl;

cout << r2 << endl;

cout << r3 << endl;

cout << r4 << endl;

cout << "please input r1:" << endl;

cin >> r1; //输入

cout << r2 << endl; //输出，格式为 1/2

cout << -r1 << endl; //一元+,－运算

cout << r1 + r3 << endl; //二元算术运算：+,-,\*,/

cout <<(r2<r1) << endl; //关系运算和逻辑运算

r4 = r1 / r2; //赋值运算和复合赋值运算

double d;

d = r4; //Rational 转换为double

cout << d << "\t" << r4 << endl;

return 0;

}