郑 州 轻 工 业 大 学

**实 验 报 告**

**课程名称： 面向对象程序设计**

**姓 名： 原彬贺**

**学 号： 542001020223**

**专业班级： 计算机科学与技术20-02**

**任课教师： 王岩**

**2021 年 11 月 3 日**

**实验报告正文**

**实验四**

**运算符重载**

**一、 实验目的**

1. 了解运算符重载的定义和使用方法；

2. 掌握运算符重载应该注意的问题；

3. 掌握常用的几种运算符的重载；

4. 理解编译时的多态性和运行时的多态性。

**二、 问题的本质和抽象描述**

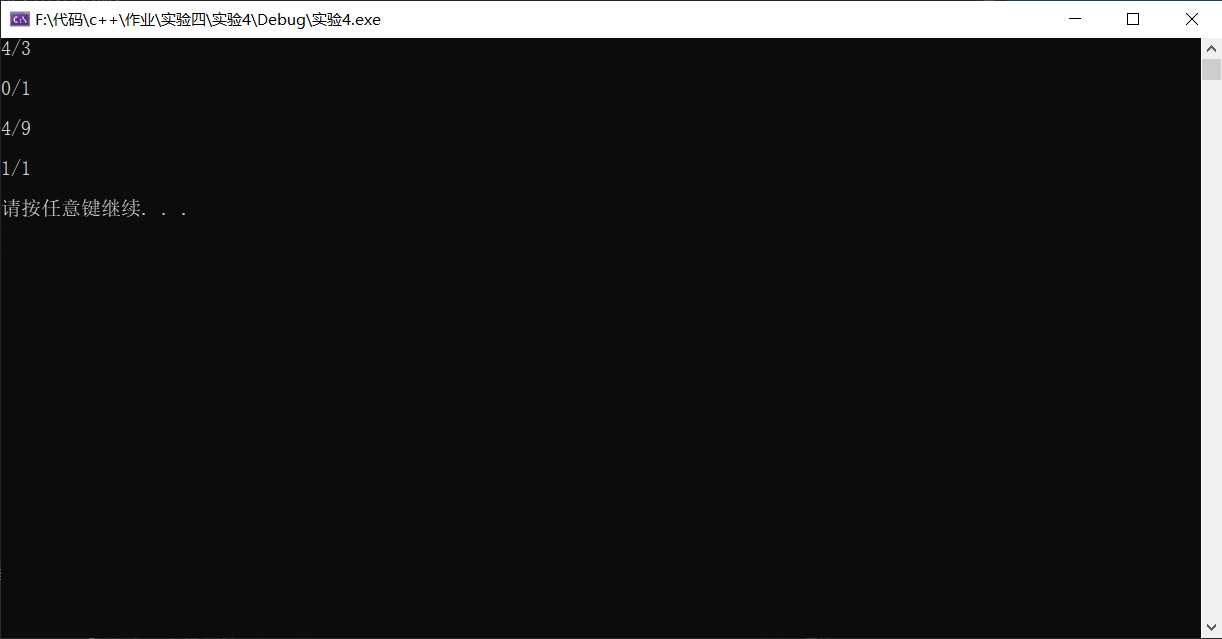
定义一个分数类，分别定义输出输入，四则运算，赋值运算符，再定义一个化简为标准型的成员函数和化简的友元函数，化简标准的友元函数时，需要先求出其最大公约数，用最大公约数将其化简。在输出与赋值时，要注意传递类时，需要常引用，防止数据丢失。

**三、 测试**

**1.方案**

先定义一个Rational的类，在其中定义分子与分母为其私有数据成员，再定义输入与输出函数为友元函数，重载输入输出运算符，在输出函数中，对类的引用应该为1常引用。再定义四则运算运算符与赋值运算符。在定义化简函数时，应当注意要将其设为友元函数，以至于在四则运算结束后，可以对其进行化简存储。

**2.结果**



**四、 总结与讨论**

在进行实验时，通过对于运算符的重载熟悉了友元函数和成员函数的细微差别，在实验时，将化简函数设置为友元函数，则可以直接将在通过传入其引用，将其化简成最简分数并进行存储，在进行输入时，需要单独定义一个字符变量来吸收分数中的除号，进行输出时便可以直接将‘/’进行输出。

**附：源代码**

#include<iostream>

using namespace std;

class Rational;

istream& operator >>(istream& istr, Rational& r);

ostream& operator <<(ostream& ostr, const Rational& d);

class Rational

{

private:

long num, den;

void standardize(void);

public:

Rational(int num = 0, int den = 1);

Rational operator+(Rational);

Rational operator-(Rational);

Rational operator\*(Rational);

Rational operator/(Rational);

Rational operator=(const Rational&);

friend Rational huajian(Rational&);

friend istream& operator>>(istream& istr, Rational&);

friend ostream& operator<<(ostream& ostr,const Rational&d);

};

Rational::Rational(int n, int m) :num(n),den(m)

{

if (den == 0)

{

cerr << "A zero denminator is invalid" << endl;

exit(1);

}

}

void Rational::standardize()

{

if (den < 0)

{

den = -den;

num = -num;

}

}

Rational huajian(Rational& t)

{

int n, m, d;

n = t.num;

m = t.den;

while (m!=0)

{

d = n % m;

n = m;

m = d;

}

t.num = t.num / n;

t.den = t.den / n;

return t;

}

istream& operator >>(istream& istr, Rational& r)

{

char c;

istr >> r.num >> c >> r.den;

if (r.den== 0)

{

cerr << "A zero denminator is invalid" << endl;

exit(1);

}

r.standardize();

return istr;

}

ostream& operator <<(ostream& ostr, const Rational& d)

{

ostr << d.num <<'/' << d.den<<endl;

return ostr;

}

Rational Rational::operator+(Rational r)

{

Rational temp;

temp.num = this->den \* r.num+r.den\*this->num;

temp.den = this->den \* r.den;

huajian(temp);

return temp;

}

Rational Rational::operator-(Rational r)

{

Rational temp;

temp.num = r.den \* this->num- this->den \* r.num;

temp.den = this->den \* r.den;

if (temp.num == 0)

{

temp.num = 0;

temp.den = 1;

return temp;

}

huajian(temp);

return temp;

}

Rational Rational::operator\*(Rational r)

{

Rational temp;

temp.num = this->num \* r.num;

temp.den = this->den \* r.den;

huajian(temp);

return temp;

}

Rational Rational::operator/(Rational r)

{

Rational temp;

temp.num = this->num \* r.den;

temp.den = this->den \* r.num;

huajian(temp);

return temp;

}

Rational Rational ::operator=(const Rational& r)

{

if (this == &r)

{

return \*this;

}

this->num = r.num;

this->den = r.den;

}

int main()

{

Rational a(2, 3),b(4,6),c;

c = a + b;

cout << c << endl;

c = a - b;

cout << c << endl;

c = a \* b;

cout << c << endl;

c = a / b;

cout << c << endl;

system("pause");

return 0;

}