**深 圳 大 学 实 验 报 告**

**课程名称： 面向对象程序设计**

**实验项目名称： 类和对象**

**学院： 电子与信息工程学院**

**专业： 电子信息工程**

**指导教师： 赵 博**

**报告人：刘付明泉 学号：2017132099班级：电子信息工程03**

**实验时间： 2019年11月7日**

**实验报告提交时间： 2019年11月18日**

**教务处制**

| 实验目的与要求：  实验目的：   1. 掌握面向对象的基本概念和类的定义方法； 2. 掌握类成员的访问权限以及访问类成员的方法； 3. 掌握引用的概念及应用，掌握引用作为函数参数的方法； 4. 掌握运算符重载为成员函数的方法； 5. 掌握运算符重载为友元函数的方法。   实验要求：   1. 建立一个分数类。分数类的成员包括分子和分母，操作包括约分、通分、加、减、乘、除、求倒数、比较、显示、输入； 2. 加、减、乘、除、比较的操作采用运算符重载的方式实现； 3. 功能要求：能够处理int型数据与分数类对象的混合运算； 4. 简述成员函数与友元函数的区别，并分析哪些功能需要用成员函数实现，哪些功能需要用友元函数实现； 5. 测试用例涵盖所有功能（包括分数类之间的运算以及分数类与int型数据的运算），需包含运行结果图、测试目的说明、运行结果分析； 6. 代码整洁、规范 |
| --- |
| 方法、步骤：  1.编写分数的类定义  2.先实现基本分数与分数的加减乘除  3.编写分数与整数的加减乘除  4.用友元函数实现整数对分数的加减乘除 |

实验过程及内容：

#include"iostream"

using namespace std;

class fraction

{

private:

int above;

int below;

void reduction(); //约分

void makeCommond(fraction); //通分

public:

fraction(int above = 0, int below = 1)

{

this->above = above;

if (below == 0)below = 1;

this->below = below;

}

friend fraction operator+(int a, fraction f0) {

fraction newf;

fraction f(a, 1);

f0.makeCommond(f);

f.makeCommond(f0);

newf.above = f0.above + f.above;

newf.below = f0.below;

return newf;

}

friend fraction operator-(int a, fraction f0) {

fraction f(a, 1);

fraction newf;

f0.makeCommond(f);

f.makeCommond(f0);

newf.above = f.above - f0.above;

newf.below = f.below;

return newf;

}

friend fraction operator\*(int a, fraction f0) {

fraction f(a, 1);

fraction newf;

newf.above = f0.above \* f.above;

newf.below = f0.below \* f.below;

newf.reduction();

return newf;

}

friend fraction operator/(int a, fraction f0) {

fraction f(a, 1);

fraction newf;

if (f.above != 0) {

newf.above = f.above \* f0.below;

newf.below = f.below \* f0.above;

return newf;

}

else {

newf.above = 0;

newf.below = 1;

cout << endl << "除数不能为0" << endl;

return newf;

}

}

friend bool operator>(int a, fraction f0) {

fraction f(a, 1);

f0.makeCommond(f);

f.makeCommond(f0);

if (f.above > f0.above)return true;

else return false;

}

friend bool operator<(int a, fraction f0) {

fraction f(a, 1);

f0.makeCommond(f);

f.makeCommond(f0);

if (f.above < f0.above)return true;

else return false;

}

friend bool operator==(int a, fraction f0) {

fraction f(a, 1);

f0.makeCommond(f);

f.makeCommond(f0);

if (f.above == f0.above)return true;

else return false;

}

fraction operator+(fraction);

fraction operator+(int);

fraction operator-(fraction);

fraction operator-(int);

fraction operator\*(fraction);

fraction operator\*(int);

fraction operator/(fraction);

fraction operator/(int);

fraction reciprocal(); //求倒数

bool operator>(fraction);

bool operator>(int);

bool operator<(fraction);

bool operator<(int);

bool operator==(fraction);

bool operator==(int);

void display(); //显示分数

void input(); //输入

};

void fraction::reduction()

{

int max, min, remander=0;

if (above >= below)

{

max = above;

min = below;

}

else

{

max = below;

min = above;

}

remander = max % min;

while (remander)

{

max = min;

min = remander;

remander = max % min;

}

//min就是最大公因数

above = above / min;

below = below / min;

}

void fraction::makeCommond(fraction f1)

{

below = below \* f1.below;//?

above = above \* f1.below;

}

fraction fraction::operator+(fraction f2)

{

fraction temp;

temp.above = f2.above \* below;

temp.below = f2.below \* below;

makeCommond(f2);

temp.above = temp.above + above;

return temp;

}

fraction fraction::operator-(fraction f2)

{

fraction temp;

temp.above = f2.above \* below;

temp.below = f2.below \* below;

makeCommond(f2);

if (temp.above > above)

{

temp.above = temp.above - above;

}

else

{

temp.above = above - temp.above;

}

return temp;

}

fraction fraction::operator\*(fraction f3)

{

fraction temp;

temp.above = f3.above \* above;

temp.below = f3.below \* below;

return temp;

}

fraction fraction::reciprocal()

{

fraction temp;

temp.above = below;

temp.below = above;

return temp;

}

fraction fraction::operator/(fraction f) {

fraction newf;

if (f.above != 0) {

newf.above = this->above \* f.below;

newf.below = this->below \* f.above;

return newf;

}

else {

newf.above = 0;

newf.below = 1;

cout << endl << "除数不能为0" << endl;

return newf;

}

}

bool fraction::operator==(fraction f4)

{

reduction();

f4.reduction();

if (below == f4.below && above == f4.above)

{

return true;

}

else

{

return false;

}

}

bool fraction::operator>(fraction f4)

{

fraction temp;

temp.above = f4.above \* below;

temp.below = f4.below \* below;

makeCommond(f4);

if (above > temp.above)

{

return true;

}

else

{

return false;

}

}

bool fraction::operator<(fraction f4)

{

fraction temp;

temp.above = f4.above \* below;

temp.below = f4.below \* below;

makeCommond(f4);

if (above < temp.above)

{

return true;

}

else

{

return false;

}

}

void fraction::display()

{

reduction();

cout << above << "/" << below << endl;

}

void fraction::input()

{

cout << "请依次输入分子和分母用空格间开" << endl;

cin >> above >> below;

}

fraction fraction::operator+(int a) {

fraction newf;

fraction f(a, 1);

makeCommond(f);

f.makeCommond(\*this);

newf.above = this->above + f.above;

newf.below = this->below;

return newf;

}

fraction fraction::operator-(int a) {

fraction f(a, 1);

fraction newf;

makeCommond(f);

f.makeCommond(\*this);

newf.above = this->above - f.above;

newf.below = this->below;

return newf;

}

fraction fraction::operator\*(int a) {

fraction f(a, 1);

fraction newf;

newf.above = this->above \* f.above;

newf.below = this->below \* f.below;

newf.reduction();

return newf;

}

fraction fraction::operator/(int a) {

fraction f(a, 1);

fraction newf;

if (f.above != 0) {

newf.above = this->above \* f.below;

newf.below = this->below \* f.above;

return newf;

}

else {

newf.above = 0;

newf.below = 1;

cout << endl << "除数不能为0" << endl;

return newf;

}

}

bool fraction::operator>(int a) {

fraction f(a, 1);

makeCommond(f);

f.makeCommond(\*this);

if (this->above > f.above)return true;

else return false;

}

bool fraction::operator<(int a) {

fraction f(a, 1);

makeCommond(f);

f.makeCommond(\*this);

if (this->above < f.above)return true;

else return false;

}

bool fraction::operator==(int a) {

fraction f(a, 1);

makeCommond(f);

f.makeCommond(\*this);

if (this->above == f.above)return true;

else return false;

}

int main()

{

fraction test1, test2, test3;

int test4=2, test5;

test1.input();

test1.display();

test2.input();

test2.display();

cout << "分数相加" << endl;

test3 = test1+test2;

test3.display();

cout << "分数相减" << endl;

test3 = test1-test2;

test3.display();

cout << "分数相乘" << endl;

test3 = test1\*test2;

test3.display();

cout << "分数相除" << endl;

test3 = test1/test2;

test3.display();

cout << "设参与计算的整数为2:" << endl;

cout << "分数加整数" << endl;

test3 = test1 + test4;

test3.display();

cout << "分数减整数" << endl;

test3 = test1 - test4;

test3.display();

cout << "分数乘整数" << endl;

test3 = test1 \* test4;

test3.display();

cout << "分数除以整数" << endl;

test3 = test1 / test4;

test3.display();

cout << "整数加分数" << endl;

test3 = test4 + test1 ;

test3.display();

cout << "整数减分数" << endl;

test3 = test4 - test1;

test3.display();

cout << "整数乘分数" << endl;

test3 = test4 \* test1;

test3.display();

cout << "整数除以分数" << endl;

test3 = test4 / test1;

test3.display();

cout << "比较大小" << endl;

cout << "分数与分数前者大于后者的判断" << endl;

if (test1 > test2)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "分数与分数前者小于后者的判断" << endl;

if (test1 < test2)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "分数与分数前者等于后者的判断" << endl;

if (test1 == test2)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "分数与整数前者大于后者的判断" << endl;

if (test1 > test4)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "分数与分数前者小于后者的判断" << endl;

if (test1 < test4)

cout << "true" << endl;

else

cout << "false" << endl;

cout << "分数与分数前者等于后者的判断" << endl;

if (test1 == test4)

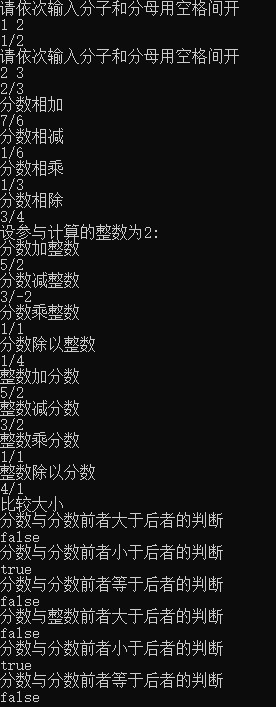
cout << "true" << endl;

else

cout << "false" << endl;

}

**结果验证截图：**



实验总结：

什么是友元函数：

    指某些虽然不是类的成员却能够访问类的所有成员的函数

为什么要使用：

    为了使其他类的成员函数来访问该类的私有变量

什么时候使用：

    可以用于运算符重载

    两个类共享数据

调用方式：

    不是类的成员函数，调用时不通过对象

存在位置：

    可以在类内声明，类外定义，可放在类的私有段或公有段，放在私有段和公有段无区别。

什么是成员函数：

    描述类的行为，是程序算法的实现部分，像构造函数、析构函数等都是类的成员函数，友元函数不是成员函数。

成员函数和友元函数区别：

相同点：

      对类的存取方式相同，可以直接存取类的任何存取控制属性的成员

      可以通过对象存取形参、函数体中该类类型对象的所有成员

不同点：

         成员函数有this指针，友元函数没有

         友元函数不能被继承，就像父亲的朋友不一定是儿子的朋友~