浙江大学实验报告

课程名称：	面向对象程序设计	实验类型：	上机
实验项目名称：	Assignment 005: Fraction		
学生姓名：	李博涵	专业：	计算机科学与技术
学号：	3130103371		
同组学生姓名：	None	指导老师：	翁恺
实验地点：	紫金港中心机房	实验日期：	2015 年 6 月 2 日

一、 实验目的和要求

To get better understand of operator overload, we here write a class presents a fraction number like 2/3.

二、 实验内容和原理

Functions below have to be implemented for this class:

- default ctor
- ctor takes two integers as parameters
- copy ctor
- arithmetical operators: + - * /
- relational operators: < <= == != >= >
- type cast to double
- toString
- inserter and extractor for streams

三、 实验平台

1、 Microsoft Windows 8.1

2、 G++

四、 操作方法与实验步骤及结果

4.1 Selected Code

Fraction.h

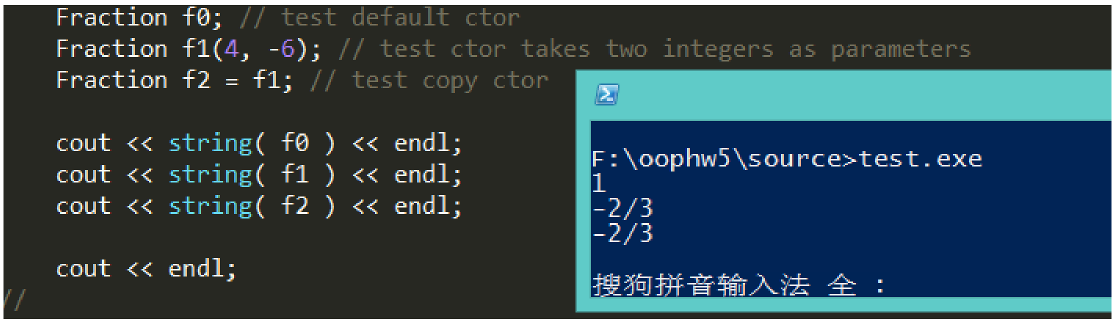
```
#ifndef __FRACTION__H__
#define __FRACTION__H__


#include <iostream>
#include <string>
#include <sstream>
#include <cstdlib>

class Fraction
{
private:
    int numerator, denominator;
public:
    // default ctor: init as 1/1
    Fraction() : numerator(1), denominator(1) { }

    // ctor takes two integers as parameters
    Fraction(int a, int b); // b == 0 is not allowed!

    // default ctor: init as 1/1
    Fraction(const Fraction &f) : numerator(f.numerator),
denominator(f.denominator) { }

    // arithmetical operators: + - * /
    const Fraction operator+(const Fraction &f) const{
        return Fraction(numerator * f.denominator + f.numerator *
denominator, denominator * f.denominator); }

    const Fraction operator-(const Fraction &f) const{
        return Fraction(numerator * f.denominator - f.numerator *
denominator, denominator * f.denominator); }

    const Fraction operator*(const Fraction &f) const{
        return Fraction(numerator * f.numerator, denominator *
f.denominator); }

    const Fraction operator/(const Fraction &f) const{
        return Fraction(numerator * f.denominator, denominator *
f.numerator); }

    // relational operators: < <= == != >= >
    bool operator<(const Fraction &f) const{
        return ( numerator * f.denominator - f.numerator * denominator <
0 ); }

    bool operator<=(const Fraction &f) const{ return !( f < *this ); }

    bool operator>=(const Fraction &f) const{ return !( *this < f ); }

    bool operator>(const Fraction &f) const{ return ( f < *this ); }

    bool operator!=(const Fraction &f) const{ return ( f < *this || *this
< f ); }

    bool operator==(const Fraction &f) const{ return !( f < *this || *this
< f ); }

    // type cast to double
    operator double() const { return ( 1.0 * numerator / denominator ); }

    // to  string
    operator std::string() const;

    // greatest common divisor
    int GCD(int a, int b);
};

// inserter and extractor for streams
std::ostream& operator<<(std::ostream &os, const Fraction &f); // should
not be define explicitly here
std::istream& operator>>(std::istream &is, Fraction &f); // input in the
format of "a/b" or just "a"

#endif
```

Fraction.cpp

```
#include "Fraction.h"

Fraction::Fraction(int a, int b)  // b == 0 is not allowed!
{
    int sig, gcd;

    sig = ( a * b > 0 ) ? 1 : -1;
    a = ( a > 0 ) ? a : -a;
    b = ( b > 0 ) ? b : -b;
    gcd = GCD(a, b);
    numerator = a / gcd * sig;
    denominator = b / gcd;
}



Fraction::operator std::string() const
{
    std::ostringstream ostr;

    if( denominator != 1 && numerator != 0)
        ostr << numerator << "/" << denominator;
    else
        ostr << numerator;
    return ostr.str();
}

// greatest common divisor
int Fraction::GCD(int a, int b)
{
    int c;

    while( b !=0 )
    {
        c = a % b;
        a = b;
        b = c;
    }

    return a;
}



std::ostream& operator<<(std::ostream &os, const Fraction &f)
{
    return os << std::string(f);
}



std::istream& operator>>(std::istream &is, Fraction &f)
{
    int a, b = 1;
    std::string str, strA, strB;
    std::istream &newIs = (is >> str);
    int slashPos;

    slashPos = str.find('/');
    if( slashPos != -1 )
    {
        strA = str.substr(0, slashPos);
        strB = str.substr(slashPos + 1, str.length());
        a = atoi(strA.c_str());
        b = atoi(strB.c_str());
    }
    else
    {
        a = atoi(str.c_str());
    }
    f = Fraction(a,b);
    return newIs;
}
```

4.2 Demo

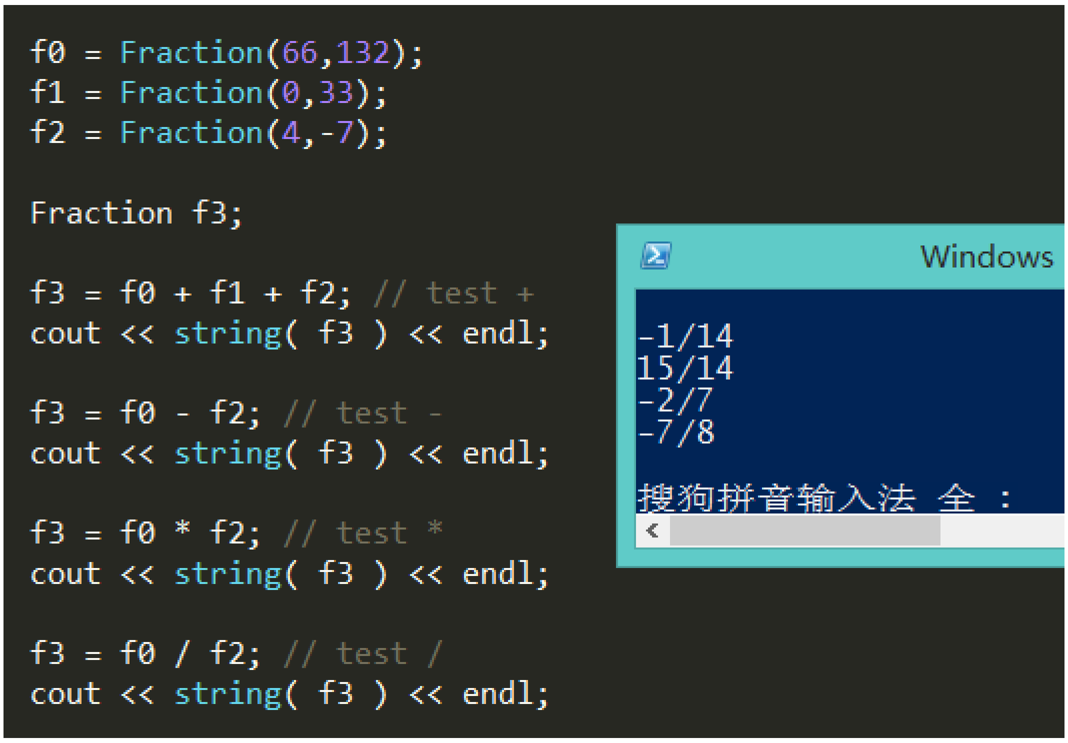
➢ **Test default ctor, ctor with parameters and copy ctor**



```
    Fraction f0; // test default ctor
    Fraction f1(4, -6); // test ctor takes two integers as parameters
    Fraction f2 = f1; // test copy ctor

    cout << string( f0 ) << endl;
    cout << string( f1 ) << endl;
    cout << string( f2 ) << endl;

    cout << endl;
//
```

```
F:\oophw5\source>test.exe
1
-2/3
-2/3

搜狗拼音输入法 全 ：
```

➢ **Test arithmetical operators**

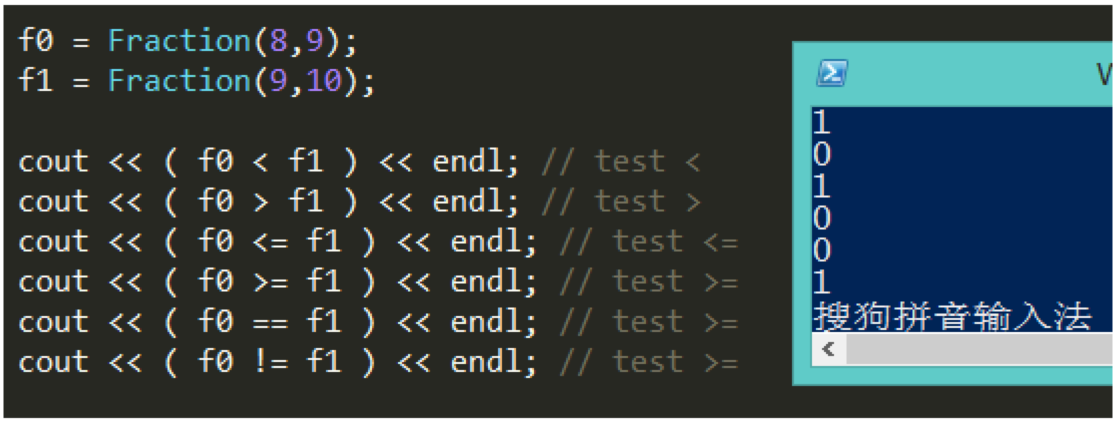


```
f0 = Fraction(66,132);
f1 = Fraction(0,33);
f2 = Fraction(4,-7);

Fraction f3;

f3 = f0 + f1 + f2; // test +
cout << string( f3 ) << endl;

f3 = f0 - f2; // test -
cout << string( f3 ) << endl;

f3 = f0 * f2; // test *
cout << string( f3 ) << endl;

f3 = f0 / f2; // test /
cout << string( f3 ) << endl;
```

```
-1/14
15/14
-2/7
-7/8
```

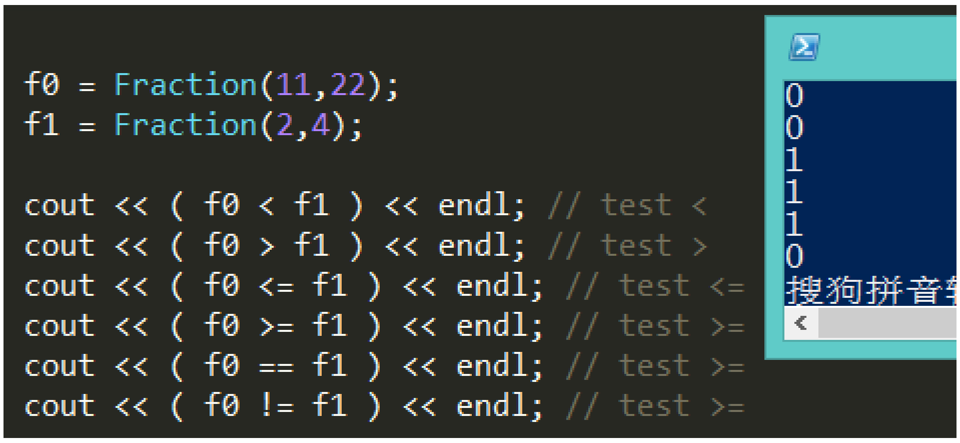
66/132 + 0/33 + 4/(-7) = -1/14
(66/132) – (4/(-7)) = 15/14
(66/132) * (4/(-7)) = -2/7
(66/132) / (4/(-7)) = -7/8

- **Test relational operators**



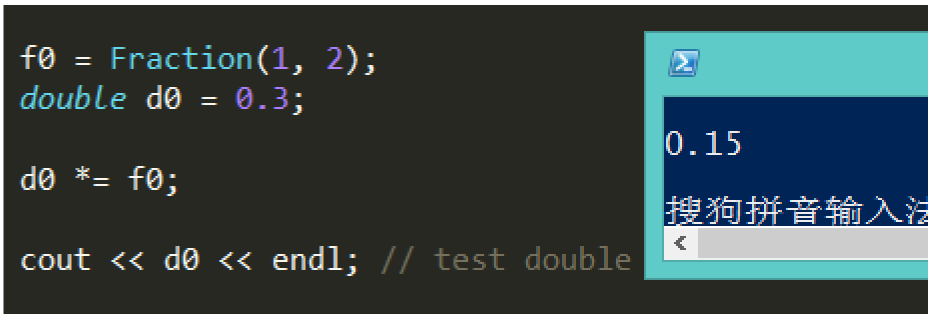
```
f0 = Fraction(8,9);
f1 = Fraction(9,10);

cout << ( f0 < f1 ) << endl; // test <
cout << ( f0 > f1 ) << endl; // test >
cout << ( f0 <= f1 ) << endl; // test <=
cout << ( f0 >= f1 ) << endl; // test >=
cout << ( f0 == f1 ) << endl; // test >=
cout << ( f0 != f1 ) << endl; // test >=
```

```
1
0
1
0
0
1
```


```
f0 = Fraction(11,22);
f1 = Fraction(2,4);

cout << ( f0 < f1 ) << endl; // test <
cout << ( f0 > f1 ) << endl; // test >
cout << ( f0 <= f1 ) << endl; // test <=
cout << ( f0 >= f1 ) << endl; // test >=
cout << ( f0 == f1 ) << endl; // test >=
cout << ( f0 != f1 ) << endl; // test >=
```

```
0
0
1
1
1
0
搜狗拼音
```

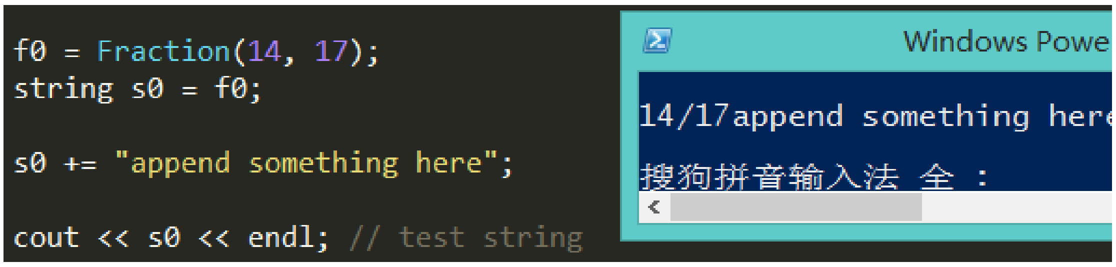
➢ **Test type cast to double**


```
f0 = Fraction(1, 2);
double d0 = 0.3;

d0 *= f0;

cout << d0 << endl; // test double
```

```
0.15
搜狗拼音输入法
```

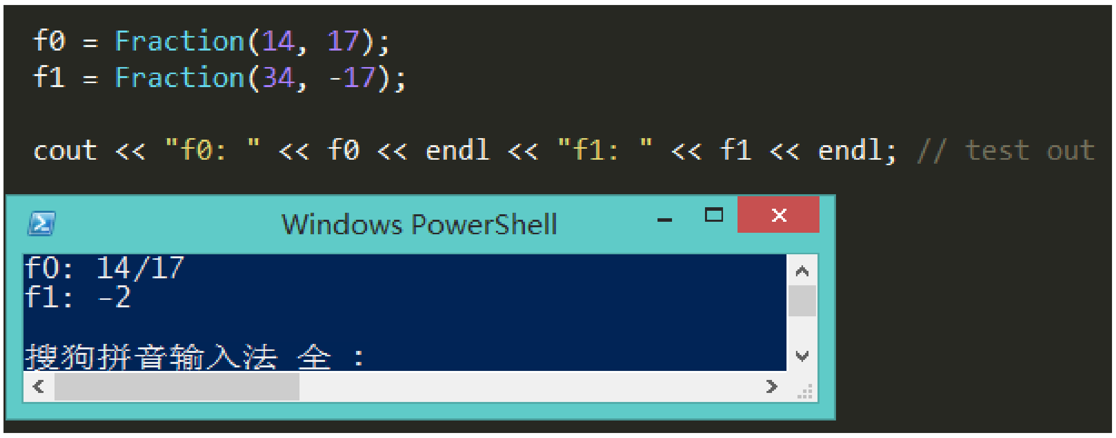
1/2 * 0.3 = 0.15

➢ **Test type cast to string**


```
f0 = Fraction(14, 17);
string s0 = f0;

s0 += "append something here";

cout << s0 << endl; // test string
```

```
14/17append something here
搜狗拼音输入法 全 :
```

➢ **inserter for streams**

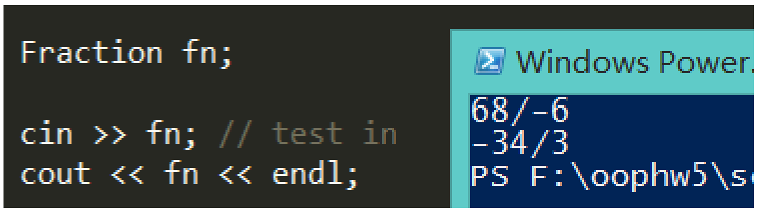


```
f0 = Fraction(14, 17);
f1 = Fraction(34, -17);

cout << "f0: " << f0 << endl << "f1: " << f1 << endl; // test out
```

Windows PowerShell

```
f0: 14/17
f1: -2

搜狗拼音输入法 全 ：
```

➢ **extractor for streams**

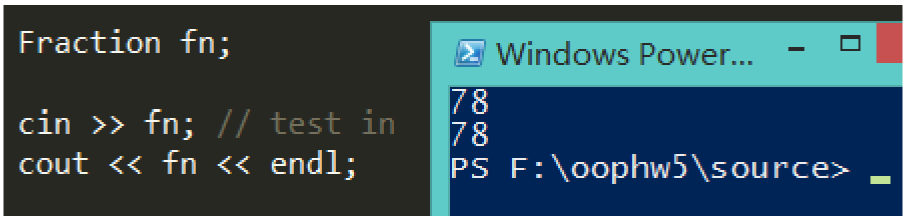


```
Fraction fn;

cin >> fn; // test in
cout << fn << endl;
```

Windows Power

```
68/-6
-34/3
PS F:\oophw5\s
```

Input 68/-6



```
Fraction fn;

cin >> fn; // test in
cout << fn << endl;
```

Windows Power...

```
78
78
PS F:\oophw5\source> _
```

Input 78