Research Project 5: Fraction

Kuan Lu

Date: 2015-6-5

Chapter 1: Introduction

The Personal Diary is a CLI (Command Line Interface) software, consists of four programs:

Chapter 2: Coding Specification

I. Class and inheritance in this project:

fraction

public:
fraction(const int a=1,const int b=1);
    ~fraction();
    fraction(const fraction& f);
    fraction operator +(const fraction& a)const;
    fraction operator -(const fraction& a)const;
    fraction operator *(const fraction& a)const;
    fraction operator /(const fraction& a)const;
    bool operator >(const fraction& a) const;
    bool operator <(const fraction& a) const;
    bool operator >=(const fraction& a) const;
    bool operator <=(const fraction& a) const;
    bool operator ==(const fraction& a) const;
    operator double() const{
        return numerator/(double)denominator;
    }
    friend istream& operator>>(istream& is,fraction& a);
    friend ostream& operator<<(ostream& os,const fraction& a);
    string toString();
private:
    int numerator;
    int denominator;

II. Source Code

(1) fraction.h

```
#ifndef FRACTION_H
#define FRACTION_H
#include <iostream>
#include <string>
#include <sstream>
```

```cpp
using namespace std;

class fraction
{
public:
   fraction(const int a=1,const int b=1);
   ~fraction();
   fraction(const fraction& f);
   fraction operator +(const fraction& a)const;
   fraction operator -(const fraction& a)const;
   fraction operator *(const fraction& a)const;
   fraction operator /(const fraction& a)const;
   bool operator >(const fraction& a) const;
   bool operator <(const fraction& a) const;
   bool operator >=(const fraction& a) const;
   bool operator <=(const fraction& a) const;
   bool operator ==(const fraction& a) const;
   operator double() const{
      return numerator/(double)denominator;
   }
   friend istream& operator>>(istream& is,fraction& a);
   friend ostream& operator<<(ostream& os,const fraction& a);
   string toString();
private:
   int numerator;
   int denominator;
};

int GreatestCommonDivisor(int x,int y);
int LeastCommonMultiple(int a, int b);

#endif // FRACTION_H
```

(2) fraction.cpp

```cpp
#include "fraction.h"

fraction::fraction(const int a,const int b)
{
   if(b==0)
     exit(0);
   int  GCD=GreatestCommonDivisor(a,b);
   if(GCD==1)
   {
        numerator=a;
        denominator=b;
    }
    else
    {
        numerator=a/GCD;
        denominator=b/GCD;
    }
}

fraction::fraction(const fraction &f)
{
   numerator=f.numerator;
   denominator=f.denominator;
}

fraction::~fraction()
{
}

fraction fraction::operator +(const fraction& a) const
{
  int tmpNumerator;
  int tmpDenominator;
  int GCD;
  int cnt1,cnt2;
  tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
  cnt1=tmpDenominator/denominator;
  cnt2=tmpDenominator/a.denominator;
  tmpNumerator=cnt1*numerator+cnt2*a.numerator;
  GCD=GreatestCommonDivisor(tmpNumerator,tmpDenominator);
  if(GCD==1)
  return fraction(tmpNumerator,tmpDenominator);
  else
  return fraction(tmpNumerator/GCD,tmpDenominator/GCD);
}

fraction fraction::operator -(const fraction& a) const
{
    int tmpNumerator;
    int tmpDenominator;
    int GCD;
    int cnt1,cnt2;
    tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
    cnt1=tmpDenominator/denominator;
    cnt2=tmpDenominator/a.denominator;
    tmpNumerator=cnt1*numerator-cnt2*a.numerator;
    GCD=GreatestCommonDivisor(tmpNumerator,tmpDenominator);
    if(GCD==1)
    return fraction(tmpNumerator,tmpDenominator);
    else
    return fraction(tmpNumerator/GCD,tmpDenominator/GCD);
}

fraction fraction::operator *(const fraction& a) const
{
   int tmpNumerator=numerator*a.numerator;
   int tmpDenominator=denominator*a.denominator;
   int GCD;
   GCD=GreatestCommonDivisor(tmpNumerator,tmpDenominator);
   if(GCD==1)
   return fraction(tmpNumerator,tmpDenominator);
   else
   return fraction(tmpNumerator/GCD,tmpDenominator/GCD);
}

fraction fraction::operator /(const fraction& a) const
{
    int tmpNumerator=numerator*a.denominator;
    int tmpDenominator=denominator*a.numerator;
    int GCD;
    GCD=GreatestCommonDivisor(tmpNumerator,tmpDenominator);
    if(GCD==1)
    return fraction(tmpNumerator,tmpDenominator);
    else
        return fraction(tmpNumerator/GCD,tmpDenominator/GCD);
}

bool fraction::operator >(const fraction &a) const
{
    int tmpDenominator;
    int cnt1,cnt2;
    tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
    cnt1=tmpDenominator/denominator;
    cnt2=tmpDenominator/a.denominator;
    if(cnt1*numerator>cnt2*a.numerator)
        return true;
    else
```

```
        return false;
}

bool fraction::operator <(const fraction &a) const
{
    int tmpDenominator;
    int cnt1,cnt2;
    tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
    cnt1=tmpDenominator/denominator;
    cnt2=tmpDenominator/a.denominator;
    if(cnt1*numerator<cnt2*a.numerator)
        return true;
    else
        return false;
}

bool fraction::operator >=(const fraction &a) const
{
    int tmpDenominator;
    int cnt1,cnt2;
    tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
    cnt1=tmpDenominator/denominator;
    cnt2=tmpDenominator/a.denominator;
    if(cnt1*numerator>=cnt2*a.numerator)
        return true;
    else
        return false;
}

bool fraction::operator <=(const fraction &a) const
{
    int tmpDenominator;
    int cnt1,cnt2;
    tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
    cnt1=tmpDenominator/denominator;
    cnt2=tmpDenominator/a.denominator;
    if(cnt1*numerator<=cnt2*a.numerator)
        return true;
    else
        return false;
}

bool fraction::operator ==(const fraction &a) const
{
    int tmpDenominator;
    int cnt1,cnt2;
    tmpDenominator=LeastCommonMultiple(denominator,a.denominator);
    cnt1=tmpDenominator/denominator;
    cnt2=tmpDenominator/a.denominator;
    if(cnt1*numerator==cnt2*a.numerator)
        return true;
    else
        return false;
}

istream &operator>>(istream &is, fraction &a)
{
   string str;
   string tmp1;
   string tmp2;
   int i=0;
   is>>str;
   while(str[i]!='/')
   {
      tmp1+=str[i];
      i++;
   }
   i++;
   int j=0;
   while(str[i])
   {
      tmp2+=str[i];
      i++;
      j++;
   }
   int x=0;
   int y=0;
   stringstream ss;
   ss<<tmp1;
   ss>>x;
   stringstream ss1;
   ss1<<tmp2;
   ss1>>y;
   fraction b(x,y);
   a=b;
   return is;
}
```

```
ostream& operator<<(ostream& os,const fraction &a)
{
    os<<a.numerator<<"/"<<a.denominator<<endl;
    return os;
}

string fraction::toString()
{
    stringstream ss;
    string str;
    ss<<numerator<<"/"<<denominator;
    ss>>str;
    return str;
}

int GreatestCommonDivisor(int x, int y)
{
   int flag=1;
   if(x<0)
   {
     flag=-flag;
     x=-x;
   }
   if(y<0)
   {
     flag=-flag;
     y=-y;
   }
   if(y == 0) return x;
   if(x < y)   return GreatestCommonDivisor(y,x)*flag;
   else       return GreatestCommonDivisor(y, x%y)*flag;
}

int LeastCommonMultiple(int a, int b)
{
    int i=1;
    int j=1;
    while(a*i!=b*j)
    {
        if(a*i<b*j)
            i++;
        else
            j++;
    }
    return a*i;
}
```

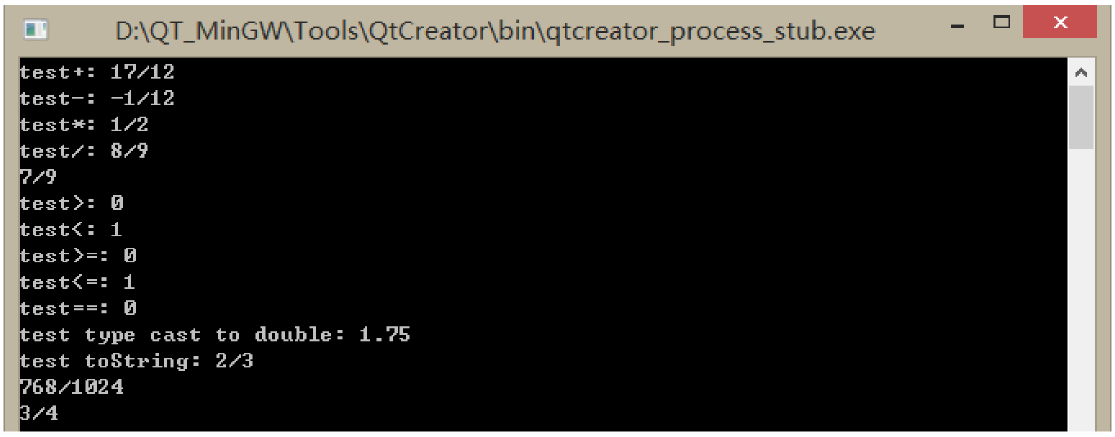
(3) main.cpp

```
#include <iostream>
#include "fraction.h"

using namespace std;

int main()
{
    fraction c;  //test default constructor
    fraction a(4,6); //test ctor with two arguments
    fraction b(3,4);
    fraction d=a;  //test copy constructor
    d=a+b;        //test +
    cout<<"test+: "<<d;      //test extractor for streams
    d=a-b;        //test -
    cout<<"test-: "<<d;
    d=a*b;        //test *
    cout<<"test*: "<<d;
    d=a/b;        //test /
    cout<<"test/: "<<d;
    cin>>c;       //test inserter for streams
    cout<<"test>: "<<(a>b)<<endl;   //test >
    cout<<"test<: "<<(a<b)<<endl;   //test <
    cout<<"test>=: "<<(a>=c)<<endl;  //test >=
    cout<<"test<=: "<<(a<=d)<<endl;  //test <=
    cout<<"test==: "<<(a==c)<<endl;  //test ==
    double e=1+b;  //test type cast to double;
    cout<<"test type cast to double: "<<e<<endl;
    cout<<"test  toString:  "<<a.toString()<<endl;  //test  function  to
string
    cin>>a;
    cout<<a<<endl;
    return 0;
}
```

Chapter 3: Test result



```
test+: 17/12
test-: -1/12
test*: 1/2
test/: 8/9
7/9
test>: 0
test<: 1
test>=: 0
test<=: 1
test==: 0
test type cast to double: 1.75
test toString: 2/3
768/1024
3/4
```

Declaration

We hereby declare that all the work done in this project titled "Personal Diary" is of my independent effort.