Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5.
Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №2
«КЛАССЫ. ПЕРЕГРУЗКА КОНСТРУКТОРОВ И ОПЕРАЦИЙ»

Выполнил:		Проверил:
студент группы ИУ5-25Б		преподаватель каф. ИУ5
Петренко Сергей		
Подпись и дата:		Подпись и дата:

г. Москва, 2018 г.

Постановка задачи:

Создать класс для работы с обыкновенными дробями.
Числитель и знаменатель дроби имеют тип int.
Дроби вводятся как строка, имеющая вид:
- для дробей с целой частью: знак, целая часть, пробел, числитель, слэш (‘/’), знаменатель
- для дробей без целой части: знак, числитель, слэш (‘/’), знаменатель.

Значения представленных выше дробей на экране при выводе должны иметь вид: -2 1/3, 8, 6. 3/4, -3, -1 1/3.
При выводе дроби сокращаются, то есть числитель и знаменатель не должны иметь общих множителей.
Перегрузить операции '+', '+=' для сложения дробей и дроби и целого в любых сочетаниях
Перегрузить операции '+', '+=' для сложения дроби и double в любых сочетаниях (дробь+double, double+дробь).
Для инициализации объектов класса обыкновенных дробей предусмотреть соответствующие конструкторы.
При перегрузке операций использовать функции - члены класса, а где это невозможно, то функции - друзья класса.
Для обеспечения более удобного контроля результатов выполнения программы вставьте в конструкторы и перегруженные операции операторы вывода, идентифицирующие выполняемую функцию.
Выполните следующий эксперимент: закомментируйте операции дроби с *int* и повторно выполните программу. Объясните результаты сложения дробей с целыми числами.

Разработка интерфейса класса:

```
class Fraction {

private:

   int numerator; // числитель

  int denominator; // знаменатель

public:

//перегрузки арифметических операций

   void operator +=( int )

   void operator +=( double)

   void  operator +=(Fraction)

   Fraction operator *(Fraction)

   Fraction operator *(int)

   Fraction  operator *(double)

   Fraction operator /(Fraction)

   Fraction operator /(int)

   Fraction  operator /(double)

   Fraction operator +(double )

   Fraction operator +(int)

   Fraction operator +(Fraction)

   Fraction operator -(double )

   Fraction operator -(int)

   Fraction operator -(Fraction)

   Fraction operator -() //Перегрузка унарного минуса

   Fraction operator = (const char* ) //перегрузка приравнивания к строке
```

void socr(int* , int*)// сокращение

Fraction(const char ps[7]) // конструктор по строке

Fraction() //путсой конструктор

Fraction(int num , int denom) // конструктор по двум параметрам

Fraction(int cel, int num , int denom) // конструктор по трем параметрам

Fraction(double res) // конструктор double

Fraction(int res) // конструктор int

Fraction(Fraction *fr) // конструктор копирования

int getNumerator() //геттер числителя

int getDenominator()//геттер знаменателя

//перегрузка friend операторов

friend Fraction operator + (double dbl, Fraction N)

friend Fraction operator / (double dbl, Fraction N)

friend bool operator>(Fraction f1, Fraction f2)

friend ostream& operator <<(ostream& os, Fraction v) //оператор вывода

friend istream& operator >> (istream &s, Fraction &v) // оператор ввода

Текст программы:

Main.cpp

```
#include <iostream>
#include "iomanip"
#include "fraction.h"
using namespace std;
int main() {
   //ввод дроби с клавиатуры
   cout<<"Введите дробь: \n";
   Fraction z;
   cin>>z;
   cout<<"z="<<z<<endl;
//проверка конструкторов
   Fraction fr1(10,14),fr2;
   cout<<"fr2="<<fr2<<endl;
   cout<<"fr1="<<fr1<<endl;
   Fraction fr="-1 4/8";
   cout<<"fr="<<fr<<endl;
   Fraction x(z),y;
   cout<<"x="<<x<<endl;
   double dbl=-1.25;
   Fraction f=dbl;
   cout<<"f="<<f<<endl;
```

```
//проверка перегруженной операции "+"
   y=x+z;
   cout<<"y="<<y<<endl;
   y+=x;
   f+=dbl/2;
   cout<<"f="<<f<<endl;
   y=x+dbl;
   cout<<"y="<<y<<endl;
   y=dbl+y;
   cout<<"y="<<y<<endl;
   y+=dbl;
   cout<<"y="<<y<<endl;
   int i=5;
   y+=i;
   cout<<"y="<<y<<endl;
   y=i+x;
   cout<<"y="<<y<<endl;
   y=x+i;
   cout<<"y="<<y<<endl;
   y+=dbl+i+x;
   cout<<"y="<<y<<endl;
Fraction f1, f2(-7),f3(-2, 2, 3),f4(8,3),f5(3.1415926);
cout<<f5;
cout<<boolalpha<<((f3*(-1.0))+f3-(0.01/(1+f1))>(-f4));
   return 0;
}
```

Planet.h

```
#ifndef LAB2_FRACTION_H
#define LAB2_FRACTION_H

#include "iostream"
#include <cstring>
#include <string.h>

using namespace std;
class Fraction {
private:
   int numerator; // числитель
   int denominator; // знаменатель
public:
   void operator +=( int );
   void operator +=( double);
   void  operator +=(Fraction);
   Fraction operator *(Fraction);
   Fraction operator *(int);
   Fraction  operator *(double);
   Fraction operator /(Fraction);
   Fraction operator /(int);
   Fraction  operator /(double);
   Fraction operator +(double );
   Fraction operator +(int);
   Fraction operator +(Fraction);
   Fraction operator -(double );
   Fraction operator -(int);
   Fraction operator -(Fraction);
   Fraction operator -( );
   Fraction operator = (const char* );
   void socr(int* , int*);
```

```
Fraction( const char ps[7]) {
    int len=0,cel,num,denom;
    string p=ps;
    string buf;
    string fract;
    int f=p.find(' ');
    if(f!= string::npos){
        buf=p.substr(0,f);
        cel=stoi(buf);
    }
    else  cel=0;
    fract=p.substr(f+1,p.find('/')-1);
    f= fract.find('/');
    buf=fract.substr(0,f);
    num=stoi(buf);
    buf=fract.substr(f+1,fract.length());
    denom=stoi(buf);
    num=(cel*denom)+num;
    socr(&num,&denom);
    this->numerator=num;
    this->denominator=denom;

}
Fraction(){
    this->numerator=0;
    this->denominator=1;
}
Fraction(int num , int denom){
    socr(&num,&denom);
    this->numerator=num;
    this->denominator=denom;
}
Fraction(int cel, int num , int denom){
    num=(cel*denom)+num;
    socr(&num,&denom);
    this->numerator=num;
    this->denominator=denom;
}
Fraction(double res){
    int num=(int)(res*1000);
    int denom = 1000;
    socr(&num,&denom);
    this->numerator=num;
    this->denominator=denom;
}
Fraction(int res){
    this->numerator=res;
    this->denominator=1;
}
Fraction(Fraction *fr) {
    int num=fr->getNumerator();
    int denom =fr->getDenominator();
    socr(&num,&denom);
    this->numerator=num;
    this->denominator=denom;
}
int getNumerator(){
    return this->numerator;
}
int getDenominator(){
    return this->denominator;
}
```

```
  friend Fraction operator + (double dbl,  Fraction N) {
     Fraction fr,N1(dbl);
fr = new Fraction((N1.getNumerator()*N.getDenominator()+N.getNumerator()*N1.getDenominator()),
(N1.getDenominator()*N.getDenominator()));
     return fr;
  }
  friend Fraction operator / (double dbl,  Fraction N) {
     Fraction fr,N1(dbl);
     fr = new Fraction(N.getNumerator()*N1.getDenominator(),N.getDenominator()*N1.getNumerator());
     return fr;
  }
   friend bool operator>(Fraction f1, Fraction f2) {
     if(f1.getNumerator()/f1.getDenominator()>f1.getNumerator()/f1.getDenominator()){
        return true;
     }
     return false;
  }
   friend ostream& operator <<(ostream& os, Fraction v)
  {
     int cel,num,dem;
     if(abs(v.numerator)>abs(v.denominator)){
        cel=v.numerator/v.denominator;
        num = abs(v.numerator%v.denominator);
        dem =  v.denominator;
        if(cel!=0){
           os<<cel<<" ";
        }
        else{
           if(num==0)
              os<<"0";
        }
        if(num!=0 && dem!=0){
           os<<num<<'/'<<dem;
        }
     }
     else os<<v.getNumerator()<<"/"<<v.getDenominator();
     return os;
  }
  friend istream& operator >> (istream &s, Fraction &v){
     int cel=0;
     char a;
     double a1;
     s>>a1;
     s>>a;
     if(a=='/'){
        v.numerator=a1;
        s>>v.denominator;
     }
     else if(a!='/'){
        cel = a1;
        v.numerator=a-48;
        s>>a;
        if(a=='/'){
           s>>v.denominator;
        }
        v.numerator=cel*v.denominator+v.numerator;
      }
      return s;
  }
};
#endif //LAB2_FRACTION_H
```

Planet.cpp

```
#include "fraction.h"
void Fraction :: operator +=(int i ){
  Fraction fr;
  Fraction N1(i);
  fr = new
Fraction((N1.getNumerator()*this->getDenominator()+this->getNumerator()*N1.getDenominator()),(N1.getDenominator()*this-
>getDenominator()));
  *this=fr;
}
void Fraction :: operator +=( double i ){
  Fraction fr;
  Fraction N1(i);
  fr = new
Fraction((N1.getNumerator()*this->getDenominator()+this->getNumerator()*N1.getDenominator()),(N1.getDenominator()*this-
>getDenominator()));
  *this=fr;
}
void Fraction :: operator +=(Fraction N1){
  Fraction fr;
  fr =  new
Fraction((N1.getNumerator()*this->getDenominator()+this->getNumerator()*N1.getDenominator()),(N1.getDenominator()*this-
>getDenominator()));
  *this=fr;
}
Fraction Fraction:: operator +(double dbl){
  Fraction fr;
  Fraction N1(dbl);
  fr = new
Fraction((N1.getNumerator()*this->getDenominator()+this->getNumerator()*N1.getDenominator()),(N1.getDenominator()*this-
>getDenominator()));
  return fr;
}
Fraction Fraction:: operator +(int i){
  Fraction fr;
  fr = new Fraction(i, this->getNumerator(), this->getDenominator());
  return fr;
}
Fraction Fraction:: operator +(Fraction N){
  Fraction fr;
  fr = new
Fraction((N.getNumerator()*this->getDenominator()-this->getNumerator()*N.getDenominator()),(N.getDenominator()*this->get
Denominator()));
  return fr;
}
Fraction Fraction:: operator -(double dbl){
  Fraction fr;
  Fraction N1(dbl);
  fr = new
Fraction((N1.getNumerator()*this->getDenominator()-this->getNumerator()*N1.getDenominator()),(N1.getDenominator()*this->
getDenominator()));
  return fr;
}
Fraction Fraction:: operator -(int i){
  Fraction fr;
  fr = new Fraction(i, this->getNumerator(), this->getDenominator());
  return fr;
}
```

```
Fraction Fraction:: operator -(Fraction N){
Fraction fr = new
Fraction((N.getNumerator()*this->getDenominator()-this->getNumerator()*N.getDenominator()),(N.getDenominator()*this->get
Denominator()));
  return fr;
}
Fraction Fraction:: operator -(){
  Fraction fr;
  fr = new Fraction(-1*this->getNumerator(),this->getDenominator());
  return fr;
}
Fraction Fraction:: operator *(Fraction N){
  Fraction fr;
  fr = new Fraction((N.getNumerator()*this->getNumerator()),(N.getDenominator()*this->getDenominator()));
  return fr;
}
Fraction Fraction:: operator *(int i){
  Fraction fr;
  fr = new Fraction((i*this->getNumerator()),(this->getDenominator()));
  return fr;
}
Fraction Fraction:: operator *(double dbl){
  Fraction fr;
  Fraction N(dbl);
  fr = new Fraction((N.getNumerator()*this->getNumerator()),(N.getDenominator()*this->getDenominator()));
  return fr;
}
Fraction Fraction:: operator /(Fraction N){
  Fraction fr;
  fr = new Fraction((N.getNumerator()*this->getDenominator()),(N.getDenominator()*this->getNumerator()));
  return fr;
}
Fraction Fraction:: operator /(int i){
  Fraction fr;
  fr = new Fraction((this->getNumerator()),(i*this->getDenominator()));
  return fr;
}
Fraction Fraction:: operator /(double dbl){
  Fraction fr;
  Fraction N(dbl);
  fr = new Fraction((this->getNumerator()*N.getDenominator()),(N.getNumerator()*this->getDenominator()));
  return fr;
}
Fraction Fraction:: operator = (const char* str)
{
  *this=new Fraction(str);
}
void Fraction:: socr(int *num , int *denom){
  for(int i=10;i>1;i--){
    if(*num%i==0&&*denom%i==0){
      *num/=i;
      *denom/=i;
      i=10;
    }
  }
}
```

Анализ данных

```
Введите дробь:
1/2
z=1/2
fr2=0/1
fr1=5/7
fr=-1/2
x=1/2
f=-1 1/4
y=0/1
f=-1 7/8
y=-3/4
y=-2
y=-3 1/4
y=1 3/4
y=5 1/2
y=5 1/2
y=9 3/4
3 141/1000false
```

```
Введите дробь:
3/2
z=1 1/2
fr2=0/1
fr1=5/7
fr=-1/2
x=1 1/2
f=-1 1/4
y=0/1
f=-1 7/8
y=1/4
y=-1/1
y=-2 1/4
y=2 3/4
y=6 1/2
y=6 1/2
y=11 3/4
3 141/1000false
```

```
-3/2
z=-1 1/2
fr2=0/1
fr1=5/7
fr=-1/2
x=-1 1/2
f=-1 1/4
y=0/1
f=-1 7/8
y=-2 3/4
y=-4
y=-5 1/4
y=-1/4
y=3 1/2
y=3 1/2
y=5 3/4
3 141/1000false
```

```
1 1/2
z=1 1/2
fr2=0/1
fr1=5/7
fr=-1/2
x=1 1/2
f=-1 1/4
y=0/1
f=-1 7/8
y=1/4
y=-1/1
y=-2 1/4
y=2 3/4
y=6 1/2
y=6 1/2
y=11 3/4
3 141/1000false
```