// comsc 200

// boli zhang

// completed

// main.cpp

// lab1

//

// Created by Jeff on 9/7/16.

// Copyright © 2016 Jeff zhang. All rights reserved.

//

#include <iostream>

#include <string>

#include "Frac.h"

using namespace std;

int main() {

//mathod 1

Frac x(3,4);

Frac y(2,3);

cout << " created x as "; //argument constructors.

x.show();

cout << "\n created y as ";

y.show();

cout <<endl;

Frac z = x; //assignment (=) operator.

cout << "\n Overload assignment operator = \n Frac z=x; created z as " << z << endl; //assignment (=) operator.

/\* //mathod 2

cout<< "created x :" << x.toString()

<< "\n created y :" << y.toString() << endl;

\*/

//mathod 3 operator

// cout<< " x assigned as y :" << x;

// << "\n created y :" << y << endl;

Frac zz(x);

cout << "\n copy constructor: \n Frac zz(x); created z: "<< zz << endl; //copy constructor: Frac x(y) that accept a Frac object as an argument.

cout << "\n Overloaded assignment operator + \n " << x <<" + "<< y <<" = "<< x+y << endl;

cout << "\n Overloaded assignment operator - \n " << x <<" - "<< y <<" = "<< x-y << endl;

cout << "\n Overloaded assignment operator \* \n " << x <<" \* "<< y <<" = "<< x\*y << endl;

cout << "\n Overloaded assignment operator / \n " << x <<" / "<< y <<" = "<< x/y << endl;

cout << "\n Overloaded assignment operator > \n " << y << " the predication " << x << " > "<<y <<" is " << ((x>y)? "True":"False") << endl;

cout << "\n Overloaded assignment operator < \n " << y << " the predication " << x << " < "<<y <<" is " << ((x<y)? "True":"False") << endl;

}

// comsc 200

// boli zhang

// completed

// Frac.h

// lab1

//

// Created by Jeff on 9/7/16.

// Copyright © 2016 Jeff zhang. All rights reserved.

//

#ifndef Frac\_h

#define Frac\_h

#include <string>

class Frac;

std::ostream &operator << (std::ostream &, const Frac &);

class Frac{

private:

long num;

long den;

public:

Frac(){

num=0;

den=1;

};

Frac(long n){

num=n;

den=1;

};

Frac(long n,long d){

num=n; den=d;

};

/\*

Frac& operator =(const Frac &x) { //assignment (=) operator.

num = x.num;

den = x.den;

}

\*/

Frac(const Frac &x){ //copy constructor: Frac x(y) that accept a Frac object as an argument.

num = x.num;

den = x.den;

};

/\*

Frac& operator = (const Frac &x){

num = x.num;

den =x.den;

}

\*/

Frac& operator + (const Frac &right){

Frac left;

left.num = num\*right.den + right.num\*den;

left.den = den\*right.den;

return left;

};

Frac& operator - (const Frac &right){

Frac left;

left.num = num\*right.den - right.num\*den;

left.den = den\*right.den;

return left;

};

Frac& operator \* (const Frac &right){

Frac left;

left.num = num\*right.num;

left.den = den\*right.den;

return left;

};

Frac& operator / (const Frac &right){

Frac left;

left.num = num\*right.den;

left.den = den\*right.num;

return left;

};

void show(){

std::cout<<num << "/" <<den ;

}

std::string toString(){

return std::to\_string(num) + "/" + std::to\_string(den);

}

// friend ostream operator << function definition

friend std::ostream &operator << (std::ostream &strm, const Frac &right){

strm << right.num << "/" << right.den;

return strm;

}

bool operator > (const Frac &x){

if(num\*x.den > den\*x.num)

return true;

else return false;

}

bool operator < (const Frac &x){

if(num\*x.den < den\*x.num)

return true;

else return false;

}

};

#endif /\* Frac\_h \*/

