OOP for operators and rational class

#include "stdafx.h"

#include "Rationals.h"

using namespace std;

#include <iostream>;

Rationals::Rationals(int n, int d)

{

setNumerator(n);

setDenominator(d);

}

void Rationals::setNumerator(int n)

{

numerator = n;

}

void Rationals::setDenominator(int d)

{

denominator = d;

}

int Rationals::getNumerator()

{

return numerator;

}

int Rationals::getDominator()

{

return denominator;

}

void Rationals::print()

{

cout << "Final value : " << numerator << "/" << denominator;

}

int Rationals::GDC(int a, int remainder)

{

if (remainder == 0)

{

return a;

}

else

{

return(GDC(remainder, a%remainder));

}

}

void Rationals::simplify(int &n, int &d)

{

int simplifyNum = 0;

if (d > n)

{

simplifyNum = GDC(d, n);

}

else if (d < n)

{

simplifyNum = GDC(n, d);

}

else

{

simplifyNum = GDC(n, d);

}

n = n / simplifyNum;

d = d / simplifyNum;

cout << "Simplified number is " << n << "/" << d << "\n";

}

Rationals Rationals::operator+(Rationals& r)

{

int n = (numerator \* r.denominator) + (r.numerator \* denominator);

int d = r.denominator \* denominator;

simplify(n, d);

Rationals rr(n, d);

return rr;

}

Rationals Rationals::operator-(Rationals& r)

{

int n = (numerator \* r.denominator) - (r.numerator \* denominator);

int d = r.denominator \* denominator;

simplify(n, d);

Rationals rr(n, d);

return rr;

}

Rationals Rationals::operator\*(Rationals& r)

{

int n = numerator \* r.denominator;

int d = denominator \* r.denominator;

simplify(n, d);

Rationals rr(n, d);

return rr;

}

Rationals Rationals::operator/(Rationals& r)

{

int n = numerator \* r.denominator;

int d = denominator \* r.numerator;

simplify(n, d);

Rationals rr(n, d);

return rr;

}

Rationals::~Rationals()

{

}

#include "stdafx.h"

#include <iostream>;

#include "Rationals.h";

using namespace std;

int main()

{

Rationals Ration1, Ration2,Ration3;

Ration1.setDenominator(8);

Ration1.setNumerator(4);

Ration2.setNumerator(10);

Ration2.setDenominator(30);

cout << "The first Rational numer is : " << Ration1.getNumerator() << "/" << Ration1.getDominator() << endl;

cout << "The first Rational numer is : " << Ration1.getNumerator() << "/" << Ration2.getDominator() << endl;

cout << "The add function: " << endl;

Ration3 = Ration1 + Ration2;

Ration3.print();

cout << " \nThe sub function: " << endl;

Ration3 = Ration1 - Ration2;

Ration3.print();

cout << "\n The multiply function: " << endl;

Ration3 = Ration1 \* Ration2;

Ration3.print();

cout << "\n The divide function: " << endl;

Ration3 = Ration1 / Ration2 ;

Ration3.print();

return 0;

}

#pragma once

class Rationals

{

public:

Rationals(int n = 1 , int d = 1);

void setNumerator(int n);

void setDenominator(int d);

int getNumerator();

int getDominator();

void print();

int GDC(int, int);

void simplify(int &, int &);

Rationals operator+(Rationals &);

Rationals operator-(Rationals &);

Rationals operator\*(Rationals &);

Rationals operator/(Rationals &);

~Rationals();

private:

int numerator;

int denominator;

};

#include "stdafx.h"

#include <iostream>

using namespace std;

template < class T >

void Swap(T &n1, T &n2)

{

T temp;

temp = n1;

n1 = n2;

n2 = temp;

}

int main()

{

int i1 = 1, i2 = 2;

cout << "Before passing data to function template.\n";

cout << "i1 = " << i1 << "\ni2 = " << i2;

Swap(i1, i2);

cout << "\n\nAfter passing data to function template.\n";

cout << "i1 = " << i1 << "\ni2 = " << i2 << "\n";

}

#include "stdafx.h"

#include "arrayy.h"

#include <iostream>;

using namespace std;

template < class T >

arrayy::arrayy(int size)

{

size = s;

arrayPtr = new int [s];

}

void showAll(const T \*array, const int count )

{

for ( int i = 0; i < count; i++ )

cout << array[ i ] << " ";

cout << endl << endl;

}

arrayy::~arrayy(void)

{

}

#pragma once

template < class T >

class arrayy

{

public:

arrayy(int = 10);

double insert();

void showAll();

double deleteNumbers();

~arrayy(void);

private:

T \*arrayPtr;

int size;

};