**Week 3**

**Exercise 1**

Create a class called **Rational** for performing arithmetic with fractions. Write a driver program to test your class.

Use integer variables to represent the **private** data of the class--the **numerator** and the **denominator**. Provide a **constructor** function that enables an object of this class to be initialized when it is declared to a non-zero values. The constructor should store the fraction in **reduced** form (i.e., the fraction 2/4 should be saved as ½)

Provide **public** member functions for each of the following:

1. set and get functions for both numerator and denominator.
2. **SetRational** to set numerator and denominator together.
3. **Add**ition of two rational numbers. The result should be stored in reduced form.
4. **Subtract**ion of two rational numbers. The result should be stored in reduced form.
5. **Mult**iplication of two rational numbers. The result should be stored in reduced form.
6. **Div**ision of two rational numbers. The result should be stored in reduced form.
7. I**sGreater** of two rational numbers, example ¼ is greater than 1/5.
8. **IsEqual** of two rational numbers, example ¼ is equal to 2/8.
9. **InBetween** to find a rational number between two rational numbers, example what is the rational number between 5/13 and 6/13, this could be 11/26.
10. **Show** rational numbers in the form **a/b** where **a** is the numerator and **b** is the denominator.

|  |  |  |
| --- | --- | --- |
| Add and Subtract | Multiply | Divide |
| A screenshot of a cell phone  Description automatically generated | A close up of a clock  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

**Utility functions (private functions):**

1. Find the **Least** **Common LC** dominator for two rational numbers, this could be used in Add and subtract function, for example 2/6+3/15 could be 10/30+6/30 rather than 30/90+18/90.
2. Find the **Greater Common Divisor** **GCD** for two rational numbers, this could be used in the Simplify function.
3. **Simplify** a rational number, example, 9/24 should be simplified to 3/7, this could be used when setting new number and other arithmetic operations (+, -, \*, /).

**Class testing**

You need to provide a testing code for each function you have implemented in the class to ensure it works without any error.

#include <iostream>

#include <math.h>

#include<time.h>

using namespace std;

class Rational {

public:

Rational(int = 1, int = 2);

//functions

void setRational(int, int);

void setNumerator(int);

void setDenominator(int);

int getNumerator() const { return numerator; };

int getDenominator() const { return denominator; };

Rational Add(Rational);

Rational Sub(Rational);

Rational Mul(Rational);

Rational Div(Rational);

bool IsGreater(Rational);

Rational IsEqual(Rational);

Rational InBetween(Rational);

void Show() const {

cout << "\tResult : " << getNumerator() << " / " << getDenominator() << endl;

};

private:

int numerator, denominator;

//utility function

int GreaterCommonDivisor(int, int); // denom1, denom2

int LeastCommon(int, int); //denom1, denom2

Rational Simplify();

};

Rational::Rational(int nt, int dt)

{

setRational(nt, dt);

}

void Rational::setRational(int n, int d)

{

setNumerator(n);

setDenominator(d);

}

void Rational::setNumerator(int n)

{

numerator = n;

}

void Rational::setDenominator(int d)

{

denominator = d;

}

Rational Rational::Add(Rational second)

{

return Rational(numerator \* second.denominator + denominator \* second.numerator,

denominator \* second.denominator).Simplify();

}

Rational Rational::Sub(Rational second)

{

return Rational(numerator \* second.denominator - denominator \* second.numerator,

denominator \* second.denominator).Simplify();

}

Rational Rational::Mul(Rational second)

{

return Rational(numerator \* second.numerator, denominator \* second.denominator).Simplify();

}

Rational Rational::Div(Rational second)

{

return Rational(numerator \* second.denominator, denominator \* second.numerator).Simplify();

}

bool Rational::IsGreater(Rational second)

{

return Rational(Sub(second)).getDenominator() > 0;

}

Rational Rational::IsEqual(Rational second)

{

int lc = LeastCommon(denominator, second.denominator);

int x = lc / denominator;

return Rational(numerator\*x,denominator\*x);

}

Rational Rational::InBetween(Rational second)

{

int lc = LeastCommon(denominator, second.denominator);

int x = lc / denominator;

int y = lc / second.denominator;

Rational temp1 = Rational(numerator \* x, denominator \* x);

Rational temp2 = Rational(second.numerator \* y, second.denominator \* y);

int interval = temp1.getNumerator() - temp2.getNumerator();

if (abs(interval) > 1)

{

srand(time(NULL));

int ResNum = rand() % abs(interval) + (interval > 0 ? temp2.getNumerator() : temp1.getNumerator());

int ResDenom = temp1.getDenominator();

return Rational(ResNum, ResDenom).Simplify();

}

else

{

int ResNum = (interval > 0 ? temp2.getNumerator()\*2 : 2\*temp1.getNumerator())+1;

int ResDenom = temp1.getDenominator()\*2;

return Rational(ResNum, ResDenom).Simplify();

}

}

//Utility Fuctions

int Rational::GreaterCommonDivisor(int d1, int d2)

{

if (d1 == 0)

return d2;

return GreaterCommonDivisor(d2 % d1, d1);

}

int Rational::LeastCommon(int d1, int d2)

{

return abs((d1 \* d2) / GreaterCommonDivisor(d1, d2));

}

Rational Rational::Simplify()

{

int LC;

LC = GreaterCommonDivisor(numerator, denominator);

return Rational(numerator / LC, denominator / LC);

}

int main()

{

Rational r1, r2, r3;

int x, y;

cout << "Rational Number 1 \n\tNumerator : ";

cin >> x;

r1.setNumerator(x);

cout << "\tDenominator : ";

cin >> y;

r1.setDenominator(y);

cout << "Rational Number 2 \n\tNumerator : ";

cin >> x;

r2.setNumerator(x);

cout << "\tDenominator : ";

cin >> y;

r2.setDenominator(y);

cout << "\nCALCULATION\n\n";

r3 = r1;

cout << "R1 + R1" << endl;

r3.Add(r2).Show();

r3 = r1;

cout << "R1 - R1" << endl;

r3.Sub(r2).Show();

r3 = r1;

cout << "R1 \* R2" << endl;

r3.Mul(r2).Show();

r3 = r1;

cout << "R1 / R2" << endl;

r3.Div(r2).Show();

r3 = r1;

cout << "Greater Value" << endl;

r3.IsGreater(r2) ? r1.Show() : r2.Show();

r3 = r1;

cout << "Equal Value" << endl;

r3.IsEqual(r2).Show();

r3 = r1;

cout << "In Between Value" << endl;

r3.InBetween(r2).Show();

cout << "\nEND OF PROGRAM\n";

}