**Lab 3-2**

**Partners: Jeremy Howell, Nhi Pham**

Fraction.h

#ifndef FRACTION\_H

#define FRACTION\_H

class Fraction

{

private:

int numer;

int denom;

public:

// Constructors

Fraction(int num, int den);

Fraction();

Fraction(const Fraction& fract);

~Fraction();

// Accessors

int getNumer() const;

int getDenom() const;

void print() const;

// Mutators

void setNumer(int num);

void setDenom(int den);

// Helping private member functions

private:

void normalize();

int gcd(int n, int m);

};

#endif

Fraction.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The implementation file fraction.cpp defining the instance \*

\* member functions and helper functions for Fraction class \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <cmath>

#include <cassert>

#include "Fraction.h"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The parameter constructor gets values for the numerator \*

\* and denominator, initializes the object, and normalizes the\*

\* value of the numerator and the denominator according to the\*

\* conditions defined in the class invariant. If denominator is 0, then set it to 1. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Fraction::Fraction(int num, int den)

{

numer = num;

if (den == 0) { denom = 1; }

else { denom = den; }

normalize();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The default constructor creates a fraction as 0/1. It does \*

\* not need validation. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Fraction::Fraction()

{

numer = 0;

denom = 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*The copy constructor creates a new fraction from an exisiting\*

\* object. It does not need normalization because the source \*

\* object is already normalized. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Fraction::Fraction(const Fraction& fract)

{

numer = fract.numer;

denom = fract.denom;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The destructor simply cleans up a fraction for recycling. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Fraction :: ~Fraction()

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The getNumer function is an accessor function returning the\*

\* numerator of the host object. It needs the const modifier \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Fraction::getNumer() const

{

return numer;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*The getDenum function is an accessor function returning the \*

\*denominator of the host object. It needs the const modifier.\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Fraction::getDenom() const

{

return denom;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The print function is an accessor function with a side \*

\* effect that displays the fraction object in the form x/y. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Fraction::print() const

{

cout << numer << '/' << denom << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*The setNumer is a mutator function the changes the numerator\*

\*of an existing object. The object needs normalization. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Fraction::setNumer(int num)

{

numer = num;

normalize();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The setDenom is a mutator function that changes denominator\*

\* of an existing object. The object needs normalization. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Fraction::setDenom(int den)

{

denom = den;

normalize();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Normalize function takes care of three fraction invariants \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Fraction::normalize()

{

// Handling a denominator of zero

if (denom == 0)

{

cout << "Invalid denomination. Need to quit." << endl;

assert(false);

}

// Changing the sign of denominator

if (denom < 0)

{

denom = -denom;

numer = -numer;

}

// Dividing numerator and denominator by gcd

int divisor = gcd(abs(numer), abs(denom));

numer = numer / divisor;

denom = denom / divisor;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The gcd function finds the greatest common divisor between \*

\* the numerator and the denominator. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Fraction::gcd(int n, int m)

{

int gcd = 1;

for (int k = 1; k <= n && k <= m; k++)

{

if (n % k == 0 && m % k == 0)

{

gcd = k;

}

}

return gcd;

}

Main.cpp

#include "Fraction.h"

#include <iostream>

using namespace std;

Fraction result;

Fraction& multiplyFract(Fraction& fr1, Fraction& fr2)

{

result.setNumer(fr1.getNumer() \* fr2.getNumer());

result.setDenom(fr1.getDenom() \* fr2.getDenom());

return result;

}

int main()

{

// Instantiation of some objects

Fraction fract1;

Fraction fract2(14, 21);

Fraction fract3(11, -8);

Fraction fract4(fract3);

Fraction fract5(2, 0);

// Printing the object

cout << "Printing four fractions after constructed: " << endl;

cout << "fract1: ";

fract1.print();

cout << "fract2: ";

fract2.print();

cout << "fract3: ";

fract3.print();

cout << "fract4: ";

fract4.print();

cout << "fract5: ";

fract5.print();

cout << endl;

// Using mutators

cout << "Changing the first two fractions and printing them:";

cout << endl;

fract1.setNumer(4);

cout << "fract1: ";

fract1.print();

fract2.setDenom(-5);

cout << "fract2: ";

fract2.print();

cout << endl;

// Using accessors

cout << "Testing the changes in two fractions:" << endl;

cout << "fract1 numerator: " << fract1.getNumer() << endl;

cout << "fract2 denominator: " << fract2.getDenom() << endl;

cout << endl;

//Test the function multiplication using two Fraction objects fract6(1,2) and fract7(2,3).

Fraction fract6(1, 2);

Fraction fract7(2, 3);

cout << "Testing the function multiplication using two Fraction objects fract6(1,2) and fract7(2,3): " << endl;

cout << "Multiplication result = ";

/\*multiplyFract(fract6, fract7).print();\*/

result = multiplyFract(fract6, fract7);

result.print();

return 0;

}