IN\_LAB

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

// Workshop 5: Member Operrator overloads

// File name : Fraction.h

// Date : Oct 11, 2017

// Author : Eunhee Kim

//

// Description

// Class definaition and function declarations

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

// header file guard

#ifndef SICT\_FRACTION\_H

#define SICT\_FRACTION\_H

// create namespace

namespace sict{

        // define the Fraction class

        class Fraction {

                // declare instance variables

                int numerator;

                int denominator;

                int greatestdivisor;

                // declare private member functions

                int max() const;

                int min() const;

                void reduce();

                int gcd() const;

        public:

                // declare public member functions

                Fraction();

                Fraction(int a, int b);

                bool isEmpty() const;

                void display() const;

                // declare the + operator overload

                Fraction operator+(const Fraction& rhs) const;

        };

}

#endif

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

// Workshop 5: Member Operrator overloads

// File name : Fraction.cpp

// Date : Oct 11, 2017

// Author : Eunhee Kim

//

// Description

// To overload an operator as a member function of a class

// type and to access the current object from within a

// member function.

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

// insert header files

#include <iostream>

#include "Fraction.h"

// continue the namespace

using namespace std;

namespace sict{

        // implement the default constructor

        Fraction::Fraction(){

                denominator = 0;

        }

        // implement the two-argument constructor

        Fraction::Fraction(int a, int b){

                if(a >= 0 && b > 0){

                        numerator = a;

                        denominator = b;

                        greatestdivisor = gcd();

                        reduce();

                }else{

                        denominator = 0;

                }

        }

        // implement the min query

        int Fraction::min() const{

                if(numerator > denominator){

                        return denominator;

                }else{

                        return numerator;

                }

        }

        // gcd returns the greatest common divisor of num and denom

    //

    int Fraction::gcd() const {

        int mn = min();  // min of numerator and denominator

        int mx = max();  // max of numerator and denominator

        int g\_c\_d = 1;

        bool found = false;

        for (int x = mn; !found && x > 0; --x) { // from mn decrement until divisor found

            if (mx % x == 0 && mn % x == 0) {

                found = true;

                g\_c\_d = x;

            }

        }

        return g\_c\_d;

    }

    // implement the reduce modifier

        void Fraction::reduce(){

                numerator /= greatestdivisor;

                denominator /= greatestdivisor;

        }

    // implement the display query

        void Fraction::display() const{

                if(isEmpty()){

                        cout << "no fraction stored";

                }else if(denominator == 1){

                        cout << numerator;

                }else{

                        cout << numerator << "/" << denominator;

                }

        }

    // implement the isEmpty query

        bool Fraction::isEmpty() const{

                if(denominator == 0){

                        return true;

                }else{

                        return false;

                }

}

    // implement the + operator

        Fraction Fraction:: operator+(const Fraction& rhs) const{

                int num;

                int denom;

                if(isEmpty() || rhs.isEmpty()){

                        return Fraction();

                }else{

                        num = (numerator \* rhs.denominator) + (denominator \* rhs.numerator);

                        denom = denominator \* rhs.denominator;

                        return Fraction(num, denom);

                }

        }

}

AT\_HOME

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

// Workshop 5: Member Operrator overloads

// File name : Fraction.h

// Date : Oct 15, 2017

// Author : Eunhee Kim

//

// Description

// Class definaition and function declarations

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

// header file guard

#ifndef SICT\_FRACTION\_H

#define SICT\_FRACTION\_H

// create namespace

namespace sict{

        // define the Fraction class

        class Fraction {

                // declare instance variables

                int numerator;

                int denominator;

                int greatestdivisor;

                // declare private member functions

                int max() const;

                int min() const;

                void reduce();

                int gcd() const;

        public:

                // declare public member functions

                Fraction();

                Fraction(int a, int b);

                bool isEmpty() const;

                void display() const;

                // declare the + operator overload

                Fraction operator+(const Fraction& rhs) const;

                Fraction operator\*(const Fraction& rhs) const;

                bool operator==(const Fraction& rhs) const;

                bool operator!=(const Fraction& rhs) const;

                Fraction operator+=(const Fraction& rhs);

        };

}

#endif

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

// Workshop 5: Member Operrator overloads

// File name : Fraction.cpp

// Date : Oct 15, 2017

// Author : Eunhee Kim

//

// Description

// To overload an operator as a member function of a class

// type and to access the current object from within a

// member function.

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

// insert header files

#include <iostream>

#include "Fraction.h"

// continue the namespace

using namespace std;

namespace sict{

        // implement the default constructor

        Fraction::Fraction(){

                denominator = 0;

        }

        // implement the two-argument constructor

        Fraction::Fraction(int a, int b){

                if(a >= 0 && b > 0){

                        numerator = a;

                        denominator = b;

                        greatestdivisor = gcd();

                        reduce();

                }else{

                        denominator = 0;

                }

        }

        // implement the max query

        int Fraction::max() const{

                if(numerator >= denominator){

                        return numerator;

                }else{

                        return denominator;

                }

        }

        // implement the min query

        int Fraction::min() const{

                if(numerator > denominator){

                        return denominator;

                }else{

                        return numerator;

                }

        }

        // gcd returns the greatest common divisor of num and denom

    //

    int Fraction::gcd() const {

        int mn = min();  // min of numerator and denominator

        int mx = max();  // max of numerator and denominator

        int g\_c\_d = 1;

        bool found = false;

        for (int x = mn; !found && x > 0; --x) { // from mn decrement until divisor found

            if (mx % x == 0 && mn % x == 0) {

                found = true;

                g\_c\_d = x;

            }

        }

        return g\_c\_d;

    }

    // implement the reduce modifier

        void Fraction::reduce(){

                numerator /= greatestdivisor;

                denominator /= greatestdivisor;

        }

    // implement the display query

        void Fraction::display() const{

                if(isEmpty()){

                        cout << "no fraction stored";

                }else if(denominator == 1){

                        cout << numerator;

                }else{

                        cout << numerator << "/" << denominator;

                }

        }

    // implement the isEmpty query

        bool Fraction::isEmpty() const{

                if(denominator == 0){

                        return true;

                }else{

                        return false;

                }

        }

    // implement the + operator

        Fraction Fraction:: operator+(const Fraction& rhs) const{

                int num;

                int denom;

                if(isEmpty() || rhs.isEmpty()){

                        return Fraction();

                }else{

                        num = (numerator \* rhs.denominator) + (denominator \* rhs.numerator);

                        denom = denominator \* rhs.denominator;

                        return Fraction(num, denom);

                }

        }

        // the \* operator

        Fraction Fraction::operator\*(const Fraction& rhs) const {

                int num;

                int denom;

                if (isEmpty() || rhs.isEmpty()){

                        return Fraction();

                }

                else {

                        num = numerator \* rhs.numerator;

                        denom = denominator \* rhs.denominator;

                        return Fraction(num, denom);

                }

        }

        // the == operator

        bool Fraction::operator==(const Fraction& rhs) const {

                if (isEmpty() || rhs.isEmpty()){

                        return false;

                }

                else {

                        if (numerator == rhs.denominator && denominator == rhs.denominator){

                                return true;

                        }

                        else {

                                return false;

                        }

                }

        }

        // the != operator

        bool Fraction::operator!=(const Fraction& rhs) const {

                if (isEmpty() || rhs.isEmpty()){

                        return false;

                }

                else {

                        if (numerator != rhs.denominator || denominator != rhs.denominator){

                                return true;

                        }

                        else {

                                return false;

                        }

                }

        }

        // the += operator

        Fraction Fraction::operator+=(const Fraction& rhs) {

                if (isEmpty() || rhs.isEmpty()){

                        return Fraction();

                }

                else {

                        numerator = (numerator \* rhs.denominator) + (denominator \* rhs.numerator);

                        denominator = denominator \* rhs.denominator;

                        greatestdivisor = gcd();

                        reduce();

                        return \*this;

                }

        }

}