Part C - Encapsulation

Member Operator Overloads

Workshop 5 (out of 10 marks - 3% of your final grade)

In this workshop, you are to overload member operators for a class.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities

- to overload an operator as a member function of a class type
- to access the current object
- to describe to your instructor what you have learned in completing this workshop

SUBMISSION POLICY

The "in-lab" section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the in-lab portion of the workshop during that period, ask your instructor for permission to complete the in-lab portion after the period. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (with a penalty; see below). The “at-home” portion of the lab is due on the day of your next scheduled workshop (23:59).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly back up your work.

Late submission penalties:

- In-lab submitted late, with at-home: Maximum of 20/50 for in-lab and Maximum of 50/50 for at home
- Workshop late for one week: in-lab, at-home and reflection must all be submitted for maximum of 50 / 100
- Workshop late for more than one week: in-lab, at-home and reflection must all be submitted for maximum of 30 / 100
- If any of in-lab, at-home or reflection is missing the mark will be zero.

FRACTION NUMBERS

A Fraction number is a number that can be represented as the ratio of one non-negative integer (numerator) over a positive integer (denominator). For example, 4/5, 3/7 and 12/2 are Fraction numbers (4,3, and 12 are numerators and 5, 7 and 2 are denominators). The following examples show some basic operations on two Fraction numbers (a/b) and (c/d):

Addition:

$$\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}$$

Multiplication:

$$\frac{a}{b}*\frac{c}{d}=\frac{ac}{bd}$$

Subtraction:

$$\frac{a}{b}-\frac{c}{d}=\frac{ad-bc}{bd}$$

Division:

$$\frac{a}{b}/\frac{c}{d}=\frac{ad}{bc}$$

Fraction numbers are common in many domains. For example, in baking, Ingredients in recipes are often listed as fractions to show the measurements (1/2 cup of flour going into a batch of cookie dough), or many commercials use Fraction numbers as statistics to get you to buy their products. For example, 4/5 dentists approve this toothpaste, or 9/10 women like this lipstick best.

FRACTION CLASS

Fraction class is designed and coded to hold information of a Fraction number (data members are the numerator and denominator). The Fraction class aims to support the mathematical operations on Fraction number same as the built-in types in C++.

The Fraction class is defined as follows:

```
class Fraction{
  private:
     int num;                  // Numerator
     int denom;                // Denominator

     int gcd();                // returns the greatest common divisor
                               // of num and denom


      int max();         // returns the maximum of num and denom
      int min();         // returns the minimum of num and denom
```

```
    public:

        void reduce();      // simplifies a Fraction number by dividing the
                            // numerator and denominator to their greatest
                            // common divisor

        Fraction();                 // default constructor
        Fraction(int n , int d=1); // construct n/d as a Fraction number

        void display() const;    // display a Fraction in output stream

        bool isEmpty() const;    // check if a Fraction number is in safe
                                 // empty state

       // TODO: define operator function prototypes here
}
```

Your main task is to complete the code of the **Fraction** class by defining the operator functions as **member functions**.

IN-LAB INSTRUCTION:

Download or clone workshop 5 from https://github.com/Seneca-244200/BTP-Workshop5

Open Workshop5/in_lab directory and view the code in **Fraction.h** and **Fraction.cpp.**

NOTE: A valid Fraction number has a non-negative numerator and a positive denominator. Default constructor initializes the object to **safe empty state** (an object with denom equals -1). The two-argument constructor also validates the parameters and sets the object to the safe empty state if the parameters are not valid.

Write the definitions and prototypes of following functions in **Fraction.cpp** and **Fraction.h** respectively (They are indicated in the files by //TODO tag):

Define **isEmpty** function as a member function, which returns true if the object is in safe empty state (an object is in the safe empty state if denominator (denom) equals -1).

Define **display** function, which sends a Fraction number to the output stream (with the “Numerator/denominator” format). This function just prints **"Invalid Fraction Object!"** in the screen if the object is in the safe empty state. In case that object denominator equals 1, it just print the numerator.

Define the operator functions for the following operators:

“+=”, “+”, “*”

The overload of the above operators should make the following code possible:

The member **operator+** : Adds two Fraction numbers and returns a Fraction number as the result. This function returns an object with the safe empty state if either of Fraction numbers (operands) is in safe empty state. It makes following code possible:

A+B (where A and B are Fraction objects)

The member **operator+=** : Adds two Fraction numbers and assigns the result to the left operand, then returns a reference to the left operand. If either of Fraction numbers (operands) is in safe empty state, it initializes the left operand to the safe empty state, then returns a reference to the left operand. It makes following code possible:

A+=B (where A and B are Fraction objects)

The member **operator*** : Multiplies two Fraction numbers and returns a Fraction number as the result. This function returns an object with the safe empty state if either of Fraction numbers (operands) is in safe empty state. It makes following code possible:

A*B (where A and B are Fraction objects)

NOTE: The reduce function which simplifies a Fraction number (for example, simplifies 2/4 to 1/2) has been already implemented. It has been used in the constructor, so whenever you create a Fraction object, it will be stored in the simplest form. Beside that, you can use it as needed.

The following code (**w5_in_lab.cpp**) will test your implementation:

```
// OOP244 Workshop 5: operators overloading
// File: w5_in_lab.cpp
// Version: 1.0
// Date: 2016/01/22
// Author: Heidar Davoudi
// Description:
// This file tests in-lab section of your workshop
////////////////////////////////////////////

#include <iostream>
#include "Fraction.h"
```

```
using namespace sict;
using namespace std;


int main(){

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

    cout << "Fraction Class Operators Test:" << endl;
    cout << "------------------------------" << endl;

    Fraction A;
    cout << "Fraction A; // ";
    cout << "A = ";
    A.display();
    cout << endl;

    Fraction B(1, 3);
    cout << "Fraction B(1, 3); // ";
    cout << "B = ";
    B.display();
    cout << endl;


    Fraction C(-5, 15);
    cout << "Fraction C(-5, 15); //";
    cout << " C = " ;
    C.display();
    cout << endl;

    Fraction D(2, 4);
    cout << "Fraction D(2, 4); //";
    cout << " D = ";
    D.display();
    cout << endl;

    Fraction E(8, 4);
    cout << "Fraction E(8, 2); //";
    cout << " E = " ;
    E.display();
    cout << endl;

    cout << endl;

    cout << "A+B equals ";
    (A+B).display();
    cout << endl;

    cout << "B+3 equals ";
    (B+3).display();
```

```
    cout << endl;

    cout << "B+D equals ";
    (B+D).display();
    cout << endl;

    cout << "(A = D = (B+=E)) equals " ;
    (A = D = (B+=E)).display();
    cout << endl;
    cout << "Now A, D, B, and E equal " ;
    A.display();
    cout << ", " ;
    D.display();
    cout << ", ";
    B.display();
    cout << ", ";
    E.display();
    cout << endl;

    return 0;
}
```

Output Example:
(Your output should exactly match the following)

```
------------------------------
Fraction Class Operators Test:
------------------------------
Fraction A; // A = Invalid Fraction Object!
Fraction B(1, 3); // B = 1/3
Fraction C(-5, 15); // C = Invalid Fraction Object!
Fraction D(2, 4); // D = 1/2
Fraction E(8, 2); // E = 2

A+B equals Invalid Fraction Object!
B+3 equals 10/3
B+D equals 5/6
(A = D = (B+=E)) equals 7/3
Now A, D, B, and E equal 7/3, 7/3, 7/3, 2
```

IN-LAB SUBMISSION

If not on matrix already, upload **Fraction.cpp, Fraction.h** and **w5_in_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 200_w5_lab <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT-HOME INSTRUCTION (40%):

Open Workshop5/at_home directory and view the code in **Fraction.h** and **Fraction.cpp.**

Define member operator functions for the following operators (add them to ones you have already implemented in the in_lab part):

"=", "!=", "operator double()", "operator int()"

Write the definitions and prototypes of the functions in **Fraction.cpp** and **Fraction.h** respectively.

The member **operator==** compares two Fraction numbers and returns true if the first operand is equal to the second one and otherwise returns false. (Hint: two Fraction number a/b and c/d are not equal if a*d == b*c). This function returns false if either of Fraction numbers (operands) is in safe empty state.

The member **operator!=** compares two Fraction numbers and returns true if the first operand is not equal to the second one and otherwise returns false. (Hint: two Fraction number a/b and c/d are not equal if a*d != b*c). This function returns false if either of Fraction numbers (operands) is in safe empty state.

The member **operator double()** returns the ratio of numerator to denominator in a floating point format (for example, returns 0.5 where the object stores 1/2). This function returns -1.0 if the Fraction number is in safe empty state. It makes following code possible:

(double) A (where A is a Fraction object)

The member **operator int()** returns the integer part of numerator to denominator ratio (for example, returns 4 where the object stores 9/2). This function returns -1 if the Fraction number is in safe empty state. It makes following code possible:

(int) A (where A is a Fraction object)

Here is the implementation file for the **w5_at_home.cpp** main module that you should use to test your implementation:

```
// OOP244 Workshop 5: operators overloading
// File: w5_in_lab.cpp
// Version: 1.0
// Date: 2016/01/22
// Author: Heidar Davoudi
// Description:
// This file tests in-lab section of your workshop
///////////////////////////////////////////

#include <iostream>
#include "Fraction.h"

using namespace sict;
using namespace std;


int main(){

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

    cout << "Fraction Class Operators Test:" << endl;
    cout << "-------------------------------" << endl;

    Fraction A;
    cout << "Fraction A; // ";
    cout << "A = ";
    A.display();
    cout << endl;

    Fraction B(1, 3);
    cout << "Fraction B(1, 3); // ";
    cout << "B = ";
    B.display();
    cout << endl;
```

```
Fraction C(-5, 15);
cout << "Fraction C(-5, 15); //";
cout << " C = " ;
C.display();
cout << endl;

Fraction D(2, 4);
cout << "Fraction D(2, 4); //";
cout << " D = ";
D.display();
cout << endl;

Fraction E(8, 4);
cout << "Fraction E(8, 2); //";
cout << " E = " ;
E.display();
cout << endl;

cout << endl;

cout << "(B*Rational(6) == E) equals ";
cout << (B*Fraction(6) == E);
cout << endl;

cout << "(A == C) equals ";
cout << (A == C);
cout << endl;

cout << "(double) B equals ";
cout.precision(3);
cout << (double) B;
cout << endl;


cout << "(int) B equals ";
cout << (int) B;
cout << endl;

cout << "S := ";

Fraction s = 0;
for(Fraction r = Fraction(1,2) ; r != Fraction(3,1);r +=Fraction(1,4)){
    r.display();
    cout  << ", " ;
    s += r;
```

```
    }

    cout << endl << "The sum of elements in S equals " ;
    s.display();
    cout << endl ;

    return 0;
}
```

Output Example:
(Your output should exactly match the following)

```
------------------------------
Fraction Class Operators Test:
------------------------------
Fraction A; // A = Invalid Fraction Object!
Fraction B(1, 3); // B = 1/3
Fraction C(-5, 15); // C = Invalid Fraction Object!
Fraction D(2, 4); // D = 1/2
Fraction E(8, 2); // E = 2

(B*Rational(6) == E) equals 1
(A == C) equals 0
(double) B equals 0.333
(int) B equals 0
S := 1/2, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4,
The sum of elements in S equals 65/4
```

REFLECTION (10%)

Please provide brief answers to the following questions in a text file named **reflect.txt.**

1. Discuss why operator+= should return a reference to Fraction.
2. In w5_in_lab.cpp (tester module), we calculated B+3 successfully. Explain how it is possible while we did not define the **operator+** function for Fraction and integer.
3. Explain what have you learned in this workshop.

AT-HOME SUBMISSION

If not on matrix already, upload **Fraction.h, Fraction.cpp** and **w5_home.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 200_w5_home <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.