**C++ Workshop – 150018**

**Homework Assignment #3**

**Operator Overloading and Strings**

**Question 1**

PART A

Define the class **Rational** that can be used to represent a rational number. (You may use the class that you created in Homework 1. Whatever is highlighted in purple is what was already assigned in Homework 1. Whatever is not highlighted is new.)

The class will have two attributes:

* Numerator
* Denominator
* Reminder – the denominator cannot have a zero nor negative value

Add the following constructors to your class:

* **Empty constructor -**  which will initialize the numerator and denominator to 1
* **Constructor** – which will get two parameters an initialize the attributes numerator and denominator according to the values of the received parameters. If the parameter for the denominator is 0, the set the denominator to be 1. Please note: a negative value cannot be stored in the denominator, so if a negative value is passed as a parameter for the denominator, then you need to change the numerator and denominator accordingly.
* **Copy constructor**

Add the following methods to your class:

* Setters for the numerator and denominator – Reminder: a denominator can’t be 0, so if the parameter that was received was 0, then you are to set it to 1. Also, as said above, the denominator can’t be a negative number, so if a negative value is received as a parameter, then you are to change the numerator and denominator accordingly.
* Getters for the numerator and denominator.
* A method to print out the number. The number should be printed out in the following format: numerator/denominator (for example, 1/2 3/4 54/56 etc) according to the values that are stored in the attributes – with reducing the fraction. When printing the number, if there is a 1 stored in the denominator, then only print the numerator.
* A method to reduce a rational number
  + The signature for the method is : void reduce();
  + The method should reduce the object that called the method

Aka for the following code segment:

Rational rat(2,4);

rat.reduce();

the numerator of rat is 1 and the denominator of rat is 2

**Note:** You will call this method at least twice, once when you are checking if two numbers are equal and once when you output the solution to a mathematical operatrion (+, -, \*, /)

PART B.

In order to use rational objects in a “natural way” you should add to the class the operators + (add), - (subtract), \* (multiply), / (divide), ++(pre and post-increment), --(post/pre-decrement), as well as the boolean operators !=, ==, <, >, <=, >=.

**Note:**

* The method for adding (+) will be similar to te methd **add** that you wrote in Homework #1. The method should reduce the sum. In other words:

Rational r1(1,6);

Rational r2(1,3);

Rational r3;

r3 = r1+r2;

r1 will remain 1/6

r2 will remain 1/3

r3 will be 1/2

* This will obviously, be the same for the rest of the mathematical operations (-, \*, /)
* For division, you can assume that they will not ask you to divide by 0. There is no need to check that the input is valid.
* The unary operators (++, --) do not reduce the rational object that called the operation. In other words:

Rational r1(2,6);

Rational r2;

r2 = r1++;

r1 is 8/6

r2 is 2/6

* The method for equality (operator==) should be similar to the **equals** method of Homework #1 except this time it should return true if the values are the same. In other words:

Rational r1(2,6);

Rational r2(1,3);

r2 == r1

returns true

**Note:** The method to test equality must called the function **reduce** twice, once for the object that called the method (\*this) and once for the object that was passed as a parameter (we shall call that object num)

Because you are required to use const whenever possible, calling the method **reduce** can cause errors. The easiest way to get around this error is to save the value of \*this in a temporary variable and also to save num in a temporary variable. You can then call reduce on those two temporary variables.

Here is the method:

bool Rational::operator==(const Rational& num) const {

Rational temp1 = \*this;

Rational temp2 = num;

temp1.reduce();

temp2.reduce();

return temp1.numerator == temp2.numerator && temp1.denominator == temp2.denominator;

}

* We can also write a method that checks inequality and once again we are looking at the actual values of the fraction. So in other words:

Rational r1(2,6);

Rational r2(1,3);

r2 != r1

would return false

**Note**: Writing **duplicate code** is considered to be a bad programming practice. Therefore, when writing the not equals method, it should call the equals method that has already been defined.

You may use the following definition for the method

bool Rational::operator != (const Rational& num) const{

return !(\*this == num);

}

Use the following main program which reads two rational numbers from the user in the format numerator/denominator. The program then prints the results of all the operations as seen in the example below.

#include"Rational.h"

#include <iostream>

using namespace std;

enum OPERATOR {

EXIT, ADD\_PRE, ADD\_POST, SUB\_PRE, SUB\_POST, ADD, SUB, MULT, DIV,

GREATER, LESS\_THAN, GREATER\_OR\_EQUAL, LESS\_OR\_EQUAL, EQUAL, NOT\_EQUAL};

void print(Rational rat1, Rational rat2, Rational rat3, char op) {

rat1.print();

cout << " " << op << " ";

rat2.print();

cout << " = ";

rat3.print();

cout << endl;

}

void print(Rational rat1, Rational rat2, const char\* op) {

rat1.print();

cout << " " << op << " ";

rat2.print();

cout << " ? ";

cout << endl;

}

void printBefore(const char\* op, Rational rat) {

cout << "x = " << op << "y;" << endl;

cout << "before operation y is: ";

rat.print();

cout << endl;

}

void printBefore(Rational rat, const char\* op) {

cout << "x = " << " y" << op << ";" << endl;

cout << "before operation y is: ";

rat.print();

cout << endl;

}

void printAfter(Rational rat1, Rational rat2) {

cout << "after operation x is: ";

rat2.print();

cout << endl << "y is: ";

rat1.print();

cout << endl;

}

int main()

{

int numerator, denominator;

char junk;

int op;

cout << "enter your choice:" << endl;

cin >> op;

Rational r2, r3;

while (op != EXIT) {

cout << "enter a rational number:" << endl;

cin >> numerator >> junk >> denominator;

Rational r1(numerator, denominator);

if (op >= 5) {

cout << "enter a rational number:" << endl;

cin >> numerator >> junk >> denominator;

r2.setNumerator(numerator);

r2.setDenominator(denominator);

}

switch (op) {

case ADD\_PRE:

printBefore("++", r1);

r3 = ++r1;

printAfter(r1, r3);

break;

case ADD\_POST:

printBefore(r1, "++");

r3 = r1++;

printAfter(r1, r3);

break;

case SUB\_PRE:

printBefore("--", r1);

r3 = --r1;

printAfter(r1, r3);

break;

case SUB\_POST:

printBefore(r1, "--");

r3 = r1--;

printAfter(r1, r3);

break;

case ADD:

r3 = r1 + r2;

print(r1, r2, r3, '+');

break;

case SUB:

r3 = r1 - r2;

print(r1, r2, r3, '-');

break;

case MULT:

r3 = r1 \* r2;

print(r1, r2, r3, '\*');

break;

case DIV:

r3 = r1 / r2;

print(r1, r2, r3, '/');

break;

case GREATER:

print(r1, r2, ">");

if (r1 > r2)

cout << "yes\n";

else cout << "no\n";

break;

case LESS\_THAN:

print(r1, r2, "<");

if (r1 < r2)

cout << "yes\n";

else cout << "no\n";

break;

case GREATER\_OR\_EQUAL:

print(r1, r2, ">=");

if (r1 >= r2)

cout << "yes\n";

else cout << "no\n";

break;

case LESS\_OR\_EQUAL:

print(r1, r2, "<=");

if (r1 <= r2)

cout << "yes\n";

else cout << "no\n";

break;

case EQUAL:

print(r1, r2, "==");

if (r1 == r2)

cout << "yes\n";

else cout << "no\n";

break;

case NOT\_EQUAL:

print(r1, r2, "!=");

if (r1 != r2)

cout << "yes\n";

else cout << "no\n";

break;

}

cout << "enter your choice : " << endl;

cin >> op;

}

return 0;

}

**QUESTION #2**

Define a class Date to represent dates. The class should contain the following fields:

* day (int)
* month (int)
* year (int)

The class should implement the following methods:

* constructor that receives values for day, month, year which defaults to the date 1/1/1920 (that is, if one or more input values are not passed, the missing parameter(s) is assigned the corresponding default value). The constructor initializes its data members from the values it receives and performs the following checks before it assigns the values:
  + - for the field day, if the value is not between 1-30 it prints the error message ERROR day and assigns it a value of 1.
    - for the field month, if the value is not between 1-12 it prints the error message ERROR month and assigns it a value of 1.
    - for the field year, if the value is less than 1920 it prints the error message ERROR year and assigns it a value of 1920.
    - Of course, it is possible to print multiple error messages for a given date.
* copy constructor
* method setDate() that updates the date. The method receives values for each field. If the values are in the legal range (as described in the constructor), then it updates its data members with the given values. If any of the values are not in the legal range, no change is made to any of the fields; however there is no error output.
* print method that prints the date in the format dd/mm/yyyy (day and month are two digits, year is four digits)
* = operator – for assignment
* pre and post auto-increment (++) operators that update the date by one day. Keep in mind that if you make it to the day after the month then you have to also update the month, if you make it to the day after the last day of the year, then you also have to update the year.

Make sure that it works correctly with the assignment operator. In other words if the program assigns a date that is being increment using the pre++ then it will assign the value that the object is after it is incremented, if the program assigns a date that it being increment post++ then it will assign the value that the object was before it increment itself.

* greater than (>) operator that checks priority of two dates. It returns true if and only if its left-hand operand is later than its right hand operand
* less than (<) operator that checks priority of two dates. It returns true if and only if its left-hand operand is earlier than its right hand operand
* equal (==) operator

For Boolean values you should print true or false.  
Note: You may assume all months are 30 days long.

Write a main program that prints Enter a date and initializes an object of type Date with the values entered. The program then prints the date (using its print method).

Next, in a loop the program prints Enter a code, reads the input (code) from the user, executes the appropriate action (described below), and prints the results.

Please note: in order to be check the ++ operation, you are to assign the variable that you are performing ++ to another variable. Upon completion, you are to print out both variables.

For example:

In order to test the ++ operation you would use the following code:

MyDate d1, d2;

d1 = ++d2;

d1.print();

cout << endl;

d2.print();

cout << endl;

The program exits when it receives a code 0.

Action codes:

1. The program prints Enter a date, inputs a new date from the user, calls setDate() to update the date, and prints the updated date.
2. The program prints the result of calling the pre auto-increment operator
3. The program prints the result of calling the post auto-increment operator
4. The program prints Enter a date, and reads in a new date into another object of type Date.

Afterwards, the program will call the operator= method and assign the object on the right side of the sign to the object on the left side of the sign. Afterwards, the program should print the object that was on the left hand side’s values.

1. The program prints Enter a date, inputs a new date from the user for a new object, and prints

> between the first date and the second date, then prints a colon (:) and then prints the result from of calling the > operator (whose left-hand operator is the original date).

1. The program prints Enter a date, inputs a new date from the user for a new object, and prints < between the first date and the second date, then prints a colon (:) and then prints the result from of calling the < operator (whose left-hand operator is the original date).
2. The program prints Enter a date, inputs a new date from the user for a new object, and prints == between the first date and the second date, then prints a colon (:) and then prints the result from of calling the == operator (whose left-hand operator is the original date).

0 The program terminates its loop and exits.

Enter a date

-5/1/2012

Error day

1/1/2012

Enter a code:

1

Enter a date

5/7/2010

5/7/2010

Enter a code:

2

6/7/2010

6/7/2010

Enter a code:

3

6/7/2010

7/7/2010

Enter a code:

4

Enter a date

14/7/2010

14/7/2010

Enter a code:

5

Enter a date

14/7/2010

14/7/2010 > 14/7/2010 : false

Enter a code:

7

Enter a date

14/7/2010

14/7/2010 == 14/7/2010 : true

Enter an a code:

0

**QUESTION #3**

Complete the definition for the class My**String** that was presented in class (examples are found in github).

a. In order to better understand the move constructor and the move assignment you should

* Add to the move constructor a statement that prints : move ctor
* Add to the class a move assignment and add to it a statement that prints : move assign

b. You should add the following methods:

* + operators <, >, <=, >=, != (according to lexicographic order)
  + the subscript operator [ ] for assigning and returning characters in the string. In the case of an error it should print ERROR and exit the program

use the command **exit(0);**

(we will soon learn a better solution.)

* + the method **insert** that takes a string and index as input and creates and returns a new string containing the contents of the current string with the given string inserted at the given index. The method returns the newly created object. In case of illegal index it should print ERROR and return an empty string.

The declaration of the method is

MyString insert(int index, const char\* str)

Write a main function that reads in two strings a and b, entered on a separate line and a number n. The program should perform the following operations:

* + - print a message showing the lexicographical relationship between a and b a<b, a=b, or a>b.
    - print the string that is returned when performing insert of a into b starting at index n. In the case that the input is not legal (it is not possible to perform the method insert), print ERROR and return an empty string.
    - input a character c and an index i and change the character at index i to c in the string just created using the operator [ ] and print the updated string.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hello  World  8  a<b  ERROR  move ctor  move assign  J  3  ERROR | Hello  World  0  a<b  move ctor  move assign  HelloWorld  -  6  HelloW-rld | Hello  World  2  a<b  move ctor  move assign  WoHellorld  ?  10  ERROR | Hello  Hello  2  a=b  move ctor  move assign  HeHellollo  h  2  Hehellollo | World  Hello  5  a>b  move ctor  move assign  HelloWorld  !  5  Hello!orld |

GOOD LUCK!