**C++ Workshop – 150018**

**Homework Assignment #3**

**Operator Overloading and Strings**

1. Complete the definition for the class Rational discussed in class (with examples in github), so that it allows for a rational objects to be used in a natural way. You should add to the class the operators + (add), - (subtract), \* (multiply), / (divide), ++(pre and post-increment), --(post/pre auto-decrement), as well as the boolean operators !=, ==, <, >, <=, >=.

If the input for the denominator of a rational is 0, you should print the message ERROR and represent the denominator as 1. When printing a number, if the denominator is 1 or the numerator is 0, output just the numerator. Always print the fraction in its simplified form.

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;

//הגדרה זו עבור קריאות התכנית בלבד

enum{add,sub,mult,divi,addPre,addPost,subPre,subPost,big,small,bigEq,smallEq,eequal,notEqual};

int main()

{

int numerator, denominator;

char tav;

int op=1;

while (op != 0) {

cout << "enter two rational numbers:" << endl;

cin >> numerator >> tav >> denominator;

Rational r1(numerator, denominator);

cin >> numerator >> tav >> denominator;

Rational r2(numerator, denominator);

switch (0) {

case add:r1.print(); cout << " + "; r2.print(); cout << " = "; (r1 + r2).print(); cout << endl;

case sub:r1.print(); cout << " - "; r2.print(); cout << " = "; (r1 - r2).print(); cout << endl;

case mult:r1.print(); cout << " \* "; r2.print(); cout << " = "; (r1 \* r2).print(); cout << endl;

case divi:r1.print(); cout << " / "; r2.print(); cout << " = "; (r1 / r2).print(); cout << endl;

case addPre:cout << "++"; cout << "("; r1.print(); cout << ") = "; (++r1).print(); cout << endl;

case addPost: cout << "("; r1.print(); cout << ")++"; cout << " = "; (r1++).print(); cout << endl;

case subPre:cout << "--"; cout << "("; r1.print(); cout << ") = "; (--r1).print(); cout << endl;

case subPost:cout << "("; r1.print(); cout << ")--"; cout << " = "; (r1--).print(); cout << endl;

case big:r1.print(); cout << " > "; r2.print(); cout << " ? "; (r1 > r2) ? cout << "yes" : cout << "no"; cout << endl;

case small:r1.print(); cout << " < "; r2.print(); cout << " ? "; (r1 < r2) ? cout << "yes" : cout << "no"; cout << endl;

case bigEq:r1.print(); cout << " >= "; r2.print(); cout << " ? "; (r1 >= r2) ? cout << "yes" : cout << "no"; cout << endl;

case smallEq:r1.print(); cout << " <= "; r2.print(); cout << " ? "; (r1 <= r2) ? cout << "yes" : cout << "no"; cout << endl;

case eequal:r1.print(); cout << " == "; r2.print(); cout << " ? "; (r1 == r2) ? cout << "yes" : cout << "no"; cout << endl;

case notEqual:r1.print(); cout << " != "; r2.print(); cout << " ? "; (r1 != r2) ? cout << "yes" : cout << "no"; cout << endl;

}

cout << "enter 1 to cont. 0 to exit." << endl;

cin >> op;

}

return 0; }

enter two rational numbers:

1/2 3/4

1/2 + 3/4 = 5/4

1/2 - 3/4 = -1/4

1/2 \* 3/4 = 3/8

1/2 / 3/4 = 2/3

++(1/2) = 3/2

(3/2)++ = 3/2

--(5/2) = 3/2

(3/2)-- = 3/2

1/2 > 3/4 ? no

1/2 < 3/4 ? yes

1/2 >= 3/4 ? no

1/2 <= 3/4 ? yes

1/2 == 3/4 ? no

1/2 != 3/4 ? yes

enter 1 to cont. 0 to exit.

0

1. 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 1/1/1970 (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 not between 1970-2099 it prints the error message ERROR year and assigns it a value of 1970.
    - 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.
* pre and post auto-increment (++) operators that update the date by one day.
* += operator that updates the date by the number of days specified by the right-hand operand
* 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
* print method that prints the date in the format dd/mm/yyyy (day and month are two digits, year is four digits)

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 What do you want to do, reads the input (code) from the user, executes the appropriate action (described below), and prints the results.

Pay attention, executing the action and printing its results (if it is a date) should be done in one statement (e.g., dateForExample++.print())

The program exits when it receives a code -1.

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 # days, inputs a number from the user, and prints the result of calling the += operator with the number received
5. The program prints Enter a date, inputs a new date from the user for a new object, and prints >: and the result from of calling the > operator (whose left-hand operator is the original date).
6. The program prints Enter a date, inputs a new date from the user for a new object, and prints <: and the result from of calling the < operator (whose left-hand operator is the original date).
7. The program prints Enter a date, inputs a new date from the user for a new object, and prints ==: and the result from of calling the == operator (whose left-hand operator is the original date).

-1 The program terminates its loop and exits.

1. Complete the definition for the class **String** that was presented in class (examples are found in github). You should add the following methods:

Enter a date

-5/1/2012//קלט מהמשתמש

Error day

1/1/2012//הדפסת הערך שהוצב בתוך התאריך

What do you want to do

1

Enter a date

5/7/2010//קלט מהמשתמש

5/7/2010//הדפסת התאריך אחרי השינוי

What do you want to do

2

6/7/2010// הדפסת אופרטור ++ תחילי על התאריך

What do you want to do

3

6/7/2010// הדפסת אופרטור ++ סופי על התאריך

What do you want to do

4

Enter # days

7

14/7/2010// הדפסת הוספת 7 ימים לתאריך

What do you want to do

5

Enter a date

14/7/2010

>: false

What do you want to do

7

Enter a date

14/7/2010

==: true

What do you want to do

-1

7

14/7/2010// הדפסת הוספת 7 ימים לתאריך

What do you want to do

5

Enter a date

14/7/2010

>: false

What do you want to do

7

Enter a date

14/7/2010

==: true

What do you want to do

-1

* 1. operators <, >, <=, >=, != (according to lexicographic order)
  2. the subscript operator [ ] for assigning and returning characters in the string. In the case of an error it should print ERROR.
  3. 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.

For example, if the string str contains “Hello” then the call str.insert(1, “World”) returns a new string containing “HWorldello”.

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