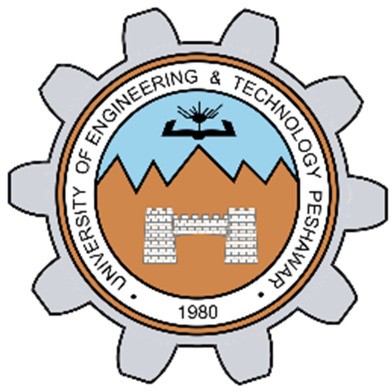
**Friend Function and Friend Class**

## LAB # 07



**Fall 2020**

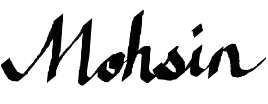
**CSE208L Object Oriented Programming Lab**

Submitted by: **Syed Mohsin Shah**

Registration No. : **19PWCSE1749**

Class Section: **A**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”



Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

## **Engr. Sumayyea Salahuddin**

January 12, 2021

Department of Computer Systems Engineering University of Engineering and Technology, Peshawar

# **Objectives of the Lab:**

* Understand the difference between a regular function and a friend function
* Explain the concept of friend function.
* Develop a friend function.
* Explain the concept of friend class.
* Develop a friend class.

Activity 1

# **Title:**

# Rational Number

# **Problem Analysis:**

Create a class RationalNumber that stores a fraction in its original form (i.e. without finding the equivalent floating pointing result). This class models a fraction by using two data members: an integer for numerator and an integer for denominator. For this class, provide the following functions:

1. A no-argument constructor that initializes the numerator and denominator of a fraction to some fixed values.
2. A two-argument constructor that initializes the numerator and denominator to the values sent from calling function. This constructor should prevent a 0 denominator in a fraction, reduce or simplify fractions that are not in reduced form, and avoid negative denominators.
3. A showRN() function to display a fraction in the format a/b.
4. Provide the following operator functions as non-member friend functions.
   1. An overloaded operator + for addition of two rational numbers.
   2. An overloaded operator - for subtraction of two rational numbers.
   3. An overloaded operator \* for subtraction of two rational numbers.
   4. An overloaded operator / for division of two rational numbers.
   5. Overloaded relational operators
      1. operator >: should return a variable of type bool to indicate whether 1st fraction is greater than 2nd or not.
      2. operator <: should return a variable of type bool to indicate whether 1st fraction is smaller than 2nd or not.
      3. operator >=: should return a variable of type bool to indicate whether 1st fraction is greater than or equal to 2nd or not.
      4. operator <=: should return a variable of type bool to indicate whether 1st fraction is smaller than or equal to 2nd or not.
   6. Overloaded equality operators for RationalNumber class
      1. operator==: should return a variable of type bool to indicate whether 1st fraction is equal to the 2nd fraction or not.
      2. operator!=: should a true value if both the fractions are not equal and return a false if both are equal.

**Algorithm:**

UML diagram for the above problem is given below:

|  |
| --- |
| **RationalNumber** |
| * num – int * denom – int |
| * RationalNumber() * RatinonalNumber(int n, int d) * showRN() * operator + (RationalNumber r1, RationalNumber r2) * operator - (RationalNumber r1, RationalNumber r2) * operator \* (RationalNumber r1, RationalNumber r2) * operator / (RationalNumber r1, RationalNumber r2) * operator > (RationalNumber r1, RationalNumber r2) * operator < (RationalNumber r1, RationalNumber r2) |

* + First make class RationalNumber:
    - Declare num and denom as private integer data members.
    - Define parameter-less Constructor method to set num = 0, denom = 1.
    - Define 2 parameter n, d Constructor method to:
      * Set num = n
      * If d>0 set denom = d else 0
      * Reduce fraction
    - Define showRN() to print values.
    - Define Operator + (r1,r2) Overloaded Function:
      * Num = (r1.num \* r2.denom) + (r1.denom \* r2.num)
      * Denom = r1.denom \* r2.denom
    - Define all other Operator Overloaded Functions.
  + In main function, create object r1, r2, input values into it, then print result.
  + Perform all operations of whose overloaded function has been defined
  + Assign result to new object r3 and Print it after each operation.

# **C++**

## **Source Code:**

#include<iostream>

using namespace std;

class RationalNumber

{

private:

int num;

int denom;

public:

RationalNumber()

{

num = 0;

denom = 1;

}

RationalNumber(int n, int d)

{

num = n;

if(d>0)

denom = d;

else

denom = 1;

//reduce fraction

for (int i = denom \* num; i > 1; i--)

{

if ((denom % i == 0) && (num % i == 0))

{

denom /= i;

num /= i;

}

}

}

void showRN()

{

cout<<"Rational Number: ";

if (denom == 1)

cout<<num<<endl;

else

cout<<num<<"/"<<denom<<endl;

}

friend RationalNumber operator + (RationalNumber const &r1, RationalNumber const &r2)

{

int n = (r1.num \* r2.denom) + (r1.denom \* r2.num);

int d = r1.denom \* r2.denom;

RationalNumber r(n,d);

return r;

}

friend RationalNumber operator - (RationalNumber const &r1, RationalNumber const &r2)

{

int n = (r1.num \* r2.denom) - (r1.denom \* r2.num);

int d = r1.denom \* r2.denom;

RationalNumber r(n,d);

return r;

}

friend RationalNumber operator \* (RationalNumber const &r1, RationalNumber const &r2)

{

int n = r1.num \* r2.num;

int d = r1.denom \* r2.denom;

RationalNumber r(n,d);

return r;

}

friend RationalNumber operator / (RationalNumber const &r1, RationalNumber const &r2)

{

int n = r1.num \* r2.denom;

int d = r1.denom \* r2.num;

RationalNumber r(n,d);

return r;

}

friend bool operator > (RationalNumber const &r1, RationalNumber const &r2)

{

float f1 = r1.num/r1.denom;

float f2 = r2.num/r2.denom;

return f1 > f2;

}

friend bool operator < (RationalNumber const &r1, RationalNumber const &r2)

{

float f1 = r1.num/r1.denom;

float f2 = r2.num/r2.denom;

return f1 < f2;

}

friend bool operator >= (RationalNumber const &r1, RationalNumber const &r2)

{

float f1 = r1.num/r1.denom;

float f2 = r2.num/r2.denom;

return f1 >= f2;

}

friend bool operator <= (RationalNumber const &r1, RationalNumber const &r2)

{

int f1 = r1.num/r1.denom;

int f2 = r2.num/r2.denom;

return f1 <= f2;

}

friend bool operator == (RationalNumber const &r1, RationalNumber const &r2)

{

int f1 = r1.num/r1.denom;

int f2 = r2.num/r2.denom;

return f1 == f2;

}

friend bool operator != (RationalNumber const &r1, RationalNumber const &r2)

{

int f1 = r1.num/r1.denom;

int f2 = r2.num/r2.denom;

return f1 != f2;

}

};

int main()

{

int n1,n2,d1,d2;

cout<<"Enter First Rational Number:";

cin>>n1>>d1;

cout<<"Enter Second Rational Number:";

cin>>n2>>d2;

RationalNumber r1(n1, d1), r2(n2, d2), r3;

cout<<"\nFractions 1 and 2:"<<endl;

r1.showRN();

r2.showRN();

cout<<"\nAddition: "<<endl;

r3 = r1 + r2;

r3.showRN();

cout<<"\nSubtraction: "<<endl;

r3 = r1 - r2;

r3.showRN();

cout<<"\nMultiplication: "<<endl;

r3 = r1 \* r2;

r3.showRN();

cout<<"\nDivision: "<<endl;

r3 = r1 / r2;

r3.showRN();

cout<<"\nIs Fraction 1 greater than Fraction 2: ";

(r1 > r2)?cout<<"True":cout<<"False";

cout<<"\nIs Fraction 1 less than Fraction 2: ";

(r1 < r2)?cout<<"True":cout<<"False";

cout<<"\nIs Fraction 1 greater or equal to Fraction 2: ";

(r1 >= r2)?cout<<"True":cout<<"False";

cout<<"\nIs Fraction 1 less or equal to Fraction 2: ";

(r1 <= r2)?cout<<"True":cout<<"False";

cout<<"\nIs Fraction 1 equal to Fraction 2: ";

(r1 == r2)?cout<<"True":cout<<"False";

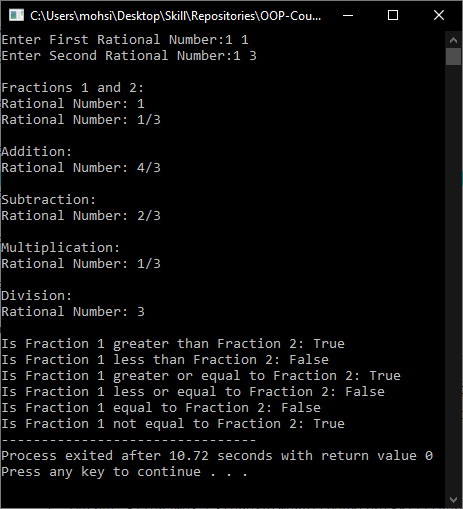
cout<<"\nIs Fraction 1 not equal to Fraction 2: ";

(r1 != r2)?cout<<"True":cout<<"False";

return 0;

}

## **Output:**



Activity 2

# **Title:**

# Multiple Inheritance

# **Problem Analysis:**

Create a class called Time that has separate int member data for hours, minutes, and seconds. Provide the following member functions for this class:

1. A no-argument constructor to initialize hour, minutes, and seconds to 0.
2. A 3-argument constructor to initialize the members to values sent from the calling function at the time of creation of an object. Make sure that valid values are provided for all the data members. In case of an invalid value, set the variable to 0.
3. A member function show to display time in 11:59:59 format.
4. Provide the following functions as friends
5. An overloaded operator+ for addition of two Time objects. Each time unit of one object must add into the corresponding time unit of the other object. Keep in view the fact that minutes and seconds of resultant should not exceed the maximum limit (60). If any of them do exceed, subtract 60 from the corresponding unit and add a 1 to the next higher unit.
6. Overloaded operators for pre- and post- increment. These increment operators should add a 1 to the seconds unit of time. Keep track that seconds should not exceed 60.
7. Overload operators for pre- and post- decrement. These decrement operators should subtract a 1 from seconds unit of time. If number of seconds goes below 0, take appropriate actions to make this value valid.

A main() programs should create two initialized Time objects and one that isn’t initialized. Then it should add the two initialized values together, leaving the result in the third Time variable. Finally it should display the value of this third variable. Check the functionalities of ++ and -- operators of this program for both pre- and post-incrementation.

**Algorithm:**

UML diagram for the above problem is given below:

|  |
| --- |
| **Time** |
| * hr – int * min – int * sec – int |
| * Time() * Time(int h, int m, int s) * show() * operator + (Time t1, Time t2) * operator ++ (Time t) * operator – (Time t) |

* + First make class Time:
    - Declare hr, min, sec as private integer data member.
    - Define parameter-less Constructor to initialize all to 0.
    - Define parameterized Contstructor:
      * hr = h, min = m that m<60, sec = s that s<60
      * All values are greater than zero
    - Define Operator Overloaded Functions
  + In main function, make t1, t2 Objects, input values into it, and print them.
  + Perform all operations of whose overloaded function has been defined.
  + Print object after each operation.

# **C++**

## **Source Code:**

#include<iostream>

using namespace std;

class Time

{

private:

int hr,min,sec;

public:

Time()

{

hr = 0;

min = 0;

sec = 0;

}

Time(int h, int m, int s)

{

if(s>=60)

{

m += (s/60);

s -= 60;

}

else if(s<0)

{

m -= 1;

s += 60;

}

if(m>=60)

{

h += (m/60);

m -= 60;

}

else if(m<0)

{

h -= 1;

m += 60;

}

hr = h;

min = m;

sec = s;

}

void show()

{

cout<<"Time:\t"<<hr<<":"<<min<<":"<<sec<<endl;

}

friend Time operator + (Time const &t1, Time const &t2)

{

int h = t1.hr + t2.hr;

int m = t1.min + t2.min;

int s = t1.sec + t2.sec;

Time t(h,m,s);

return t;

}

friend Time operator ++ (Time const &t1)

{

int s = t1.sec + 1;

Time t(t1.hr,t1.min,s);

return t;

}

friend Time operator ++ (Time const &t1, int i)

{

int s = t1.sec + 1;

Time t(t1.hr,t1.min,s);

return t;

}

friend Time operator -- (Time const &t1)

{

int s = t1.sec + 1;

Time t(t1.hr,t1.min,s);

return t;

}

friend Time operator -- (Time const &t1, int i)

{

int s = t1.sec + 1;

Time t(t1.hr,t1.min,s);

return t;

}

};

int main()

{

Time t1(4,30,50), t2(9,45,30);

cout<<"Both Times: "<<endl;

t1.show();

t2.show();

cout<<"\nAdded Time: "<<endl;

Time t3 = t1 + t2;

t3.show();

cout<<"\nPre-Incremented Time: "<<endl;

t1 = ++t1;

t1.show();

cout<<"\nPost-Incremented Time: "<<endl;

t1 = t1++;

t1.show();

cout<<"\nPre-Decremented Time: "<<endl;

t2 = --t2;

t2.show();

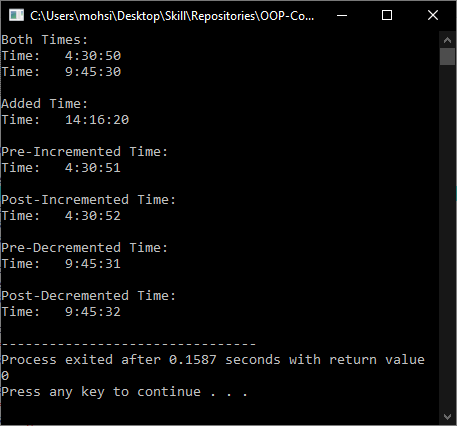
cout<<"\nPost-Decremented Time: "<<endl;

t2 = t2--;

t2.show();

}

## **Output:**



**Conclusion:**

These programs helps us in building the fundamental concepts of Friend Functions and Friend Classes in C++. It teaches us the prerequisite fundamentals for higher level programming. We learn the various concepts about the Friend Functions and Classes in OOP with the help of these programs.