**Operator Overloading**

**LAB #** **13**

**Fall 2019**

**CSE208L Object Oriented Programming Lab**

Submitted by: **Shah Raza**

Registration No. : **18PWCSE1658**

Class Section: **B**

“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 17, 2020

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Objectives of the Lab:**

Objectives of the lab are to:

# Understand operator overloading of binary and unary operators.

# Develop operator overloaded function for different functions.

* Use operators with class objects.

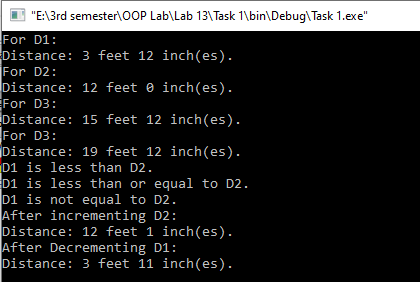
# Activity # 01

**Title:**

Create a class for Distance and overload different operators.

**In C++**

**Source code: Output:**

#include <iostream>

using namespace std;

class Distance

{

private:

int feet,inch;

public:

Distance();

Distance(int f,int i);

void ShowDistance();

Distance operator+(Distance obj);

Distance operator+=(Distance obj);

bool operator>(Distance obj);

bool operator>=(Distance obj);

bool operator<(Distance obj);

bool operator<=(Distance obj);

bool operator==(Distance obj);

bool operator!=(Distance obj);

Distance operator++();

Distance operator++(int NotUsed);

Distance operator--();

Distance operator--(int NotUsed);

};

Distance::Distance(): feet(0),inch(0){}

Distance::Distance(int f, int i): feet(f),inch(i)

{

if (inch>12)

{

feet++;

inch-=12;

}

}

void Distance::ShowDistance()

{

cout<<"Distance: "<<feet<<" feet "<<inch<<" inch(es).\n";

}

Distance Distance::operator+(Distance obj)

{

Distance temp;

temp.inch=inch+obj.inch;

temp.feet=feet+obj.feet;

if (temp.inch>12)

{

temp.feet++;

temp.inch-=12;

}

return temp;

}

Distance Distance::operator+=(Distance obj)

{

inch+=obj.inch;

feet+=obj.feet;

if (inch>12)

{

feet++;

inch-=12;

}

return \*this;

}

bool Distance::operator>(Distance obj)

{

if (feet>obj.feet)

return true;

else if(feet==obj.feet && inch>obj.inch)

return true;

else

return false;

}

bool Distance::operator<(Distance obj)

{

if (feet<obj.feet)

return true;

else if(feet==obj.feet && inch<obj.inch)

return true;

else

return false;

}

bool Distance::operator>=(Distance obj)

{

if (feet>obj.feet || (feet==obj.feet && inch==obj.inch))

return true;

else if(feet==obj.feet && inch>obj.inch)

return true;

else

return false;

}

bool Distance::operator<=(Distance obj)

{

if (feet<obj.feet || (feet==obj.feet && inch==obj.inch))

return true;

else if(feet==obj.feet && inch<obj.inch)

return true;

else

return false;

}

bool Distance::operator==(Distance obj)

{

if (feet==obj.feet && inch==obj.inch)

return true;

else

return false;

}

bool Distance::operator!=(Distance obj)

{

if (feet!=obj.feet || inch!=obj.inch)

return true;

else

return false;

}

Distance Distance::operator++()

{

inch++;

if (inch>12)

{

feet++;

inch-=12;

}

return \*this;

}

Distance Distance::operator++(int NotUsed)

{

Distance temp= \*this;

inch++;

if (inch>12)

{

feet++;

inch-=12;

}

return temp;

}

Distance Distance::operator--()

{

inch--;

if (inch<0)

{

feet--;

inch+=12;

}

return \*this;

}

Distance Distance::operator--(int NotUsed)

{

Distance temp= \*this;

inch--;

if (inch<0)

{

feet--;

inch+=12;

}

return temp;

}

int main()

{

Distance d1(3,12),d2(12,0);

cout<<"For D1: \n";

d1.ShowDistance();

cout<<"For D2: \n";

d2.ShowDistance();

Distance d3=d1+d2;

cout<<"For D3: \n";

d3.ShowDistance();

d3+=d1;

cout<<"For D3: \n";

d3.ShowDistance();

if (d1>d2)

cout<<"D1 is greater than D2.\n";

if (d1<d2)

cout<<"D1 is less than D2.\n";

if (d1>=d2)

cout<<"D1 is greater than or equal to D2.\n";

if (d1<=d2)

cout<<"D1 is less than or equal to D2.\n";

if (d1==d2)

cout<<"D1 is equal to D2.\n";

if (d1!=d2)

cout<<"D1 is not equal to D2.\n";

++d2;

cout<<"After incrementing D2:\n";

d2.ShowDistance();

d1--;

cout<<"After Decrementing D1: \n";

d1.ShowDistance();

return 0;

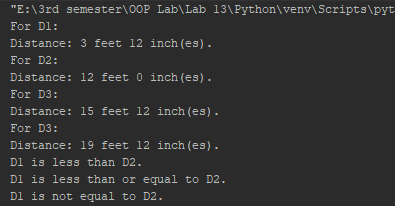
}

**In Python**

**Source code:**

class Distance:  
 def \_\_init\_\_(self,f=0,i=0):  
 self.feet=f  
 self.inch=i  
 if self.inch>12:  
 self.feet+=1  
 self.inch-=12  
 def ShowDistance(self):  
 print("Distance: {} feet {} inch(es).".format(self.feet,self.inch))  
 def \_\_add\_\_(self, other):  
 temp=Distance()  
 temp.inch=self.inch+other.inch  
 temp.feet=self.feet+other.feet  
 if temp.inch>12:  
 temp.feet+=1  
 temp.inch-=12  
 return temp  
 def \_\_iadd\_\_(self, other):  
 self.inch+=other.inch  
 self.feet+=other.feet  
 if self.inch>12:  
 self.feet+=1  
 self.inch-=12  
 return self  
 def \_\_gt\_\_(self, other):  
 if self.feet>other.feet:  
 return True  
 elif self.feet==other.feet and self.inch>other.inch:  
 return True  
 else:  
 return False  
 def \_\_lt\_\_(self, other):  
 if self.feet<other.feet:  
 return True  
 elif self.feet==other.feet and self.inch<other.inch:  
 return True  
 else:  
 return False  
 def \_\_ge\_\_(self, other):  
 if self.feet>other.feet or (self.feet==other.feet and self.inch==other.inch):  
 return True  
 elif self.feet==other.feet and self.inch>other.inch:  
 return True  
 else:  
 return False  
 def \_\_le\_\_(self, other):  
 if self.feet<other.feet or (self.feet==other.feet and self.inch==other.inch):  
 return True  
 elif self.feet==other.feet and self.inch<other.inch:  
 return True  
 else:  
 return False  
 def \_\_eq\_\_(self, other):  
 if self.feet==other.feet and self.inch==other.inch:  
 return True  
 else:  
 return False  
 def \_\_ne\_\_(self, other):  
 if self.feet!=other.feet or self.inch!=other.inch:  
 return True  
 else:  
 return False  
  
d1 = Distance(3,12)  
d2 = Distance(12,0)  
print("For D1: ")  
d1.ShowDistance()  
print("For D2: ")  
d2.ShowDistance()  
d3 = Distance()  
d3 = d1+d2  
print("For D3: ")  
d3.ShowDistance()  
d3+=d1  
print("For D3: ")  
d3.ShowDistance()  
if d1>d2:  
 print("D1 is greater than D2.")  
if d1<d2:  
 print("D1 is less than D2.")  
if d1>=d2:  
 print("D1 is greater than or equal to D2.")  
if d1<=d2:  
 print("D1 is less than or equal to D2.")  
if d1==d2:  
 print("D1 is equal to D2.")  
if d1!=d2:  
 print("D1 is not equal to D2.")

**Output:**



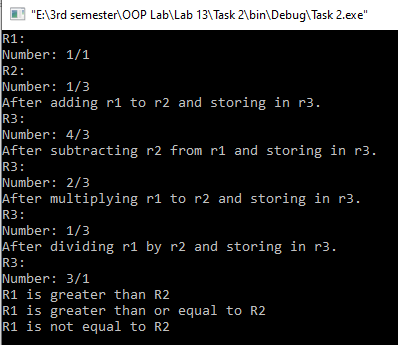
# Activity # 02

**Title:**

Create a class for RationalNumber and overload different operators.

**In C++**

**Source code: Output:**

#include <iostream>

using namespace std;

class RationalNumber

{

private:

int num;

int den;

public:

RationalNumber();

RationalNumber(int a, int b);

void ShowRN();

RationalNumber operator+(RationalNumber obj);

RationalNumber operator-(RationalNumber obj);

RationalNumber operator\*(RationalNumber obj);

RationalNumber operator/(RationalNumber obj);

bool operator>(RationalNumber obj);

bool operator<(RationalNumber obj);

bool operator>=(RationalNumber obj);

bool operator<=(RationalNumber obj);

bool operator==(RationalNumber obj);

bool operator!=(RationalNumber obj);

};

RationalNumber::RationalNumber(): num(0),den(1){}

RationalNumber::RationalNumber(int a, int b)

{

if(b<=0)

cout<<"Denominator can not be 0 or negative.";

else

{

num=a,den=b;

}

int n=num,d=den;

while(n!=d)

{

if(n>d)

n-=d;

else

d-=n;

}

num/=n;

den/=n;

}

void RationalNumber::ShowRN()

{

cout<<"Number: "<<num<<"/"<<den<<endl;

}

RationalNumber RationalNumber::operator+(RationalNumber obj)

{

RationalNumber temp;

temp.num = (num\*obj.den)+ (den\*obj.num);

temp.den = den\*obj.den;

return temp;

}

RationalNumber RationalNumber::operator-(RationalNumber obj)

{

RationalNumber temp;

temp.num = (num\*obj.den)- (den\*obj.num);

temp.den = den\*obj.den;

return temp;

}

RationalNumber RationalNumber::operator\*(RationalNumber obj)

{

RationalNumber temp;

temp.num = num\*obj.num;

temp.den = den\*obj.den;

return temp;

}

RationalNumber RationalNumber::operator/(RationalNumber obj)

{

RationalNumber temp;

temp.num = num\*obj.den;

temp.den = den\*obj.num;

return temp;

}

bool RationalNumber::operator>(RationalNumber obj)

{

if (num/den>obj.num/obj.den)

return true;

else

return false;

}

bool RationalNumber::operator<(RationalNumber obj)

{

if (num/den<obj.num/obj.den)

return true;

else

return false;

}

bool RationalNumber::operator>=(RationalNumber obj)

{

if (num/den>=obj.num/obj.den)

return true;

else

return false;

}

bool RationalNumber::operator<=(RationalNumber obj)

{

if (num/den<=obj.num/obj.den)

return true;

else

return false;

}

bool RationalNumber::operator==(RationalNumber obj)

{

if (num/den==obj.num/obj.den)

return true;

else

return false;

}

bool RationalNumber::operator!=(RationalNumber obj)

{

if (num/den!=obj.num/obj.den)

return true;

else

return false;

}

int main()

{

RationalNumber r1(1,1),r2(1,3);

cout<<"R1: \n";

r1.ShowRN();

cout<<"R2: \n";

r2.ShowRN();

RationalNumber r3=r1+r2;

cout<<"After adding r1 to r2 and storing in r3.\n";

cout<<"R3: \n";

r3.ShowRN();

r3=r1-r2;

cout<<"After subtracting r2 from r1 and storing in r3.\n";

cout<<"R3: \n";

r3.ShowRN();

r3=r1\*r2;

cout<<"After multiplying r1 to r2 and storing in r3.\n";

cout<<"R3: \n";

r3.ShowRN();

r3=r1/r2;

cout<<"After dividing r1 by r2 and storing in r3.\n";

cout<<"R3: \n";

r3.ShowRN();

if(r1>r2)

cout<<"R1 is greater than R2\n";

if(r1<r2)

cout<<"R1 is less than R2\n";

if(r1>=r2)

cout<<"R1 is greater than or equal to R2\n";

if(r1<=r2)

cout<<"R1 is less than or equal to R2\n";

if(r1==r2)

cout<<"R1 is equal to R2\n";

if(r1!=r2)

cout<<"R1 is not equal to R2\n";

return 0;

}

**In Python**

**Source code:**

class RationalNumber:  
 def \_\_init\_\_(self,a=0,b=0):  
 if b<=0:  
 print("Denominator can not be 0 or Negative.")  
 else:  
 self.num = a  
 self.den = b  
 n = self.num  
 d = self.den  
 while n!=d:  
 if n>d:  
 n-=d  
 else:  
 d-=n  
 self.num/=n  
 self.den/=n  
 def ShowRN(self):  
 print("Number: {}/{}".format(self.num,self.den))  
 def \_\_add\_\_(self, other):  
 temp = RationalNumber(1,1)  
 temp.num = (self.num \* other.den) + (self.den \* other.num)  
 temp.den = self.den \* other.den  
 return temp  
 def \_\_sub\_\_(self, other):  
 temp = RationalNumber(1,1)  
 temp.num = (self.num \* other.den) - (self.den \* other.num)  
 temp.den = self.den \* other.den  
 return temp  
 def \_\_mul\_\_(self, other):  
 temp = RationalNumber(1,1)  
 temp.num = self.num \* other.num  
 temp.den = self.den \* other.den  
 return temp  
 def \_\_div\_\_(self,other):  
 temp = RationalNumber(1,1)  
 temp.num = self.num \* other.den  
 temp.den = self.den \* other.num  
 return temp  
 def \_\_gt\_\_(self, other):  
 if self.num / self.den > other.num / other.den:  
 return True  
 else:  
 return False  
 def \_\_lt\_\_(self, other):  
 if self.num / self.den < other.num / other.den:  
 return True  
 else:  
 return False  
 def \_\_ge\_\_(self, other):  
 if self.num / self.den >= other.num / other.den:  
 return True  
 else:  
 return False  
 def \_\_le\_\_(self, other):  
 if self.num / self.den <= other.num / other.den:  
 return True  
 else:  
 return False  
 def \_\_eq\_\_(self, other):  
 if self.num / self.den == other.num / other.den:  
 return True  
 else:  
 return False  
 def \_\_ne\_\_(self, other):  
 if self.num / self.den != other.num / other.den:  
 return True  
 else:  
 return False  
  
r1 = RationalNumber(1,1)  
r2 = RationalNumber(1,3)  
print("R1: ")  
r1.ShowRN()  
print("R2: ")  
r2.ShowRN()  
r3 = RationalNumber(1,1)  
r3 = r1 + r2  
print("After adding r1 to r2 and storing in r3.")  
print("R3: ")  
r3.ShowRN()  
r3 = r1 - r2  
print("After subtracting r2 from r1 and storing in r3.")  
print("R3: ")  
r3.ShowRN()  
r3 = r1 \* r2  
print("After multiplying r1 to r2 and storing in r3.")  
print("R3: ")  
r3.ShowRN()  
if r1 > r2:  
 print("R1 is greater than R2.")  
if r1 < r2:  
 print("R1 is less than R2.")  
if r1 >= r2:  
 print("R1 is greater than or equal to R2.")  
if r1 <= r2:  
 print("R1 is less than or equal to R2.")  
if r1 == r2:  
 print("R1 is equal to R2.")  
if r1 != r2:  
 print("R1 is not equal to R2.")

**Output:**

