**Object Programming Essentials [Part 2]**

**LAB #** **02**

**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**

October 21, 2019

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Objectives of the Lab:**

Objectives of the lab are to:

# Understand how class object can be passed and returned from class member function

# Write a class with member function having objects as arguments

# Write a class with member function that return object

# Test member function effectively using given test cases

# Activity # 01

**Title:**

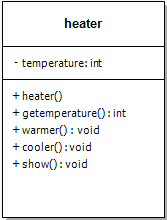
Make a class for Complex numbers and model it using real and imaginary part.

**Problem analysis:**

Create a class, **Complex** that contains two integer field, **re** and **im**. Define a constructor that takes no parameters. The **Complex** field should be set to the value 0 in the constructor. Define the mutators **subcom,** **sumcom, negcom** and **mulcom**, whose effect is to subtract, add, negate and multiply the complex numbers respectively. Define an accessor method to return the value of **Complex**. Demonstrate the use of Complex class.

**Algorithm:**

UML diagram for the above problem is given below:

* First make class complex

**Complex**

* Declare re and im as private integer field
* Define no argument constructor to set value of re and im to 0

-re,im:double

* Define display method to return value of temperature

+Complex()

+sumcom(): void

+subcom(): void

+negcom(): Complex

+mulcom(): void

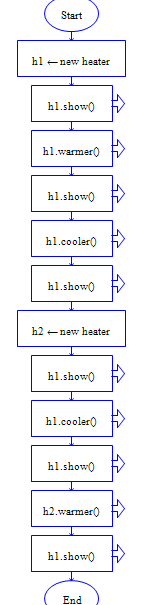
+display(): void

+input(): void

* Define sumcom, subcom, negcom and mulcom method to add, subtract, negate and multiply complex numbers respectively
* Define input function to take input from user
* In main function, make objects of complex to demonstrate theeuse of complex
* Call each function one after the other and display the show function as shown in the flow chart.

**Flowchart:**

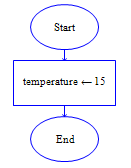
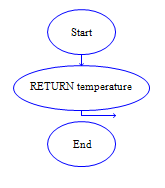
Start



C1,c2,c🡨Complex

Private: double re,im

Private: int temperature



c.sumcom(c1,c2)

Re,im🡨0

C1.display()

Cin>>re>>im

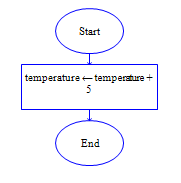
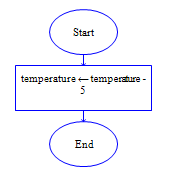
C2.display()

Void input()

Complex()

c.display()

C=c1.negcom



c.display()

C1.re-c2.re;

C1.im-c2.im;

C1.re+c2.re;

C1.im+c2.im

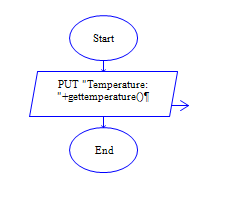
C1.input()

c.subcom(c1,c2)

Void subcom()

Void sumcom()

c.display()



c.mul(c1,c2)

Cout<<re<<im;

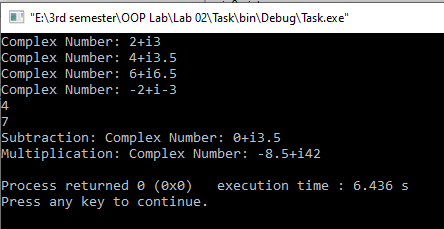
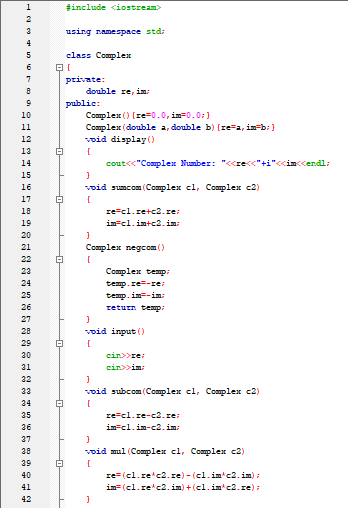
c.display()

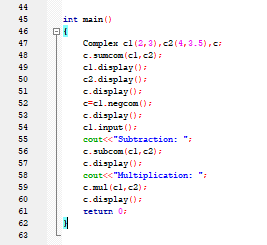
void display()

End

**In C++**

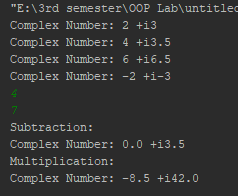
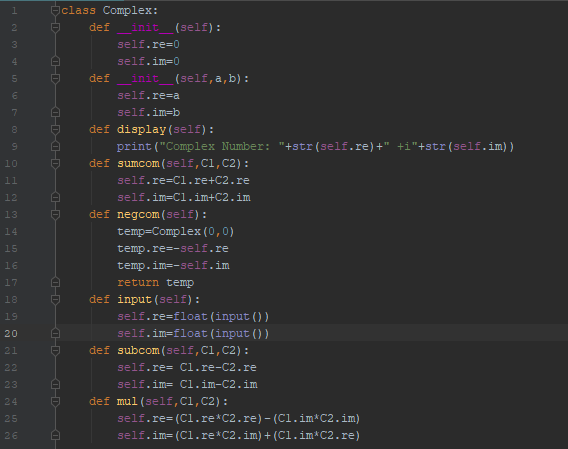
**Source code: Output:**

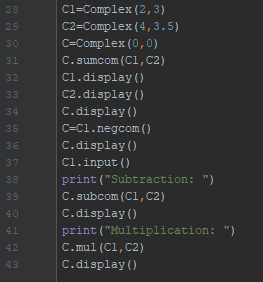
 



**In Python**

**Source code: Output:**



**Conclusion:**

This program helps us in understanding the basic concepts of classes and objects in different languages. It acts as a base for us and helps us in preparing ourselves for the higher level of programming. We get to know about the constructor and method in OOP with the help of this program.

# Activity # 02

**Title:**

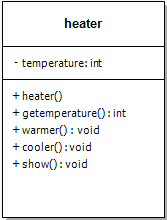
Make a Class for Complex and model it using re and im. (Return from the functions)

**Problem analysis:**

Create a class, **Complex** that contains two integer field, **re** and **im**. Define a constructor that takes no parameters. The **Complex** field should be set to the value 0 in the constructor. Define the mutators **subcom,** **sumcom, negcom** and **mulcom**, whose effect is to subtract, add, negate and multiply the complex numbers respectively. Define an accessor method to return the value of **Complex**. Demonstrate the use of Complex class.

**Algorithm:**

UML diagram for the above problem is given below:

* First make class complex

**Complex**

* Declare re and im as private integer field
* Define no argument constructor to set value of re and im to 0

-re,im:double

* Define display method to return value of temperature

+Complex()

+sumcom(): Complex

+subcom(): Complex

+negcom(): Complex

+mulcom(): Complex

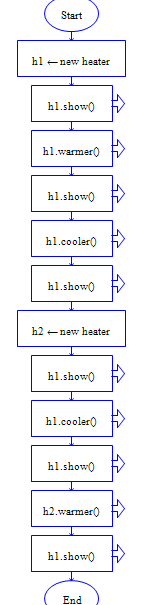
+display(): void

+input(): void

* Define sumcom, subcom, negcom and mulcom method to add, subtract, negate and multiply complex numbers respectively
* Define input function to take input from user
* In main function, make objects of complex to demonstrate theeuse of complex
* Call each function one after the other and display the show function as shown in the flow chart.

**Flowchart:**

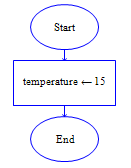
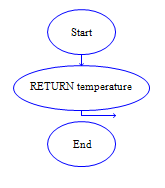
Start



C1,c2,c🡨Complex

Private: double re,im

Private: int temperature



C2=c.sumcom(c1)

Re,im🡨0

C1.display()

Cin>>re>>im

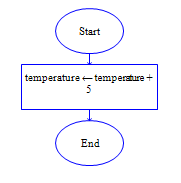
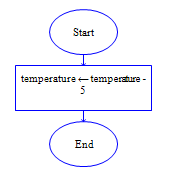
C2.display()

Void input()

Complex()

c.display()

C=c1.negcom



c.display()

C1.re-c2.re;

C1.im-c2.im;

C1.re+c2.re;

C1.im+c2.im

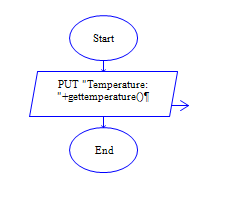
C1.input()

C=c2.subcom(c1)

Complex subcom()

Complex sumcom()

c.display()



C=C2.mul(c1)

Cout<<re<<im;

c.display()

void display()

End

**In C++**

**Source code:**

#include <iostream>

using namespace std;

class Complex

{

private:

double re,im;

public:

Complex(){re=0.0,im=0.0;}

Complex(double a,double b){re=a,im=b;}

void display()

{

cout<<"ComplexNumber: "<<re<<"+i"<<im<<endl;

}

Complex sumcom(Complex c1)

{

Complex c;

c.re=c1.re+re;

c.im=c1.im+im;

return c;

}

Complex negcom()

{

Complex temp;

temp.re=-re;

temp.im=-im;

return temp;

}

void input()

{

cin>>re;

cin>>im;

}

Complex subcom(Complex c1)

{

Complex c;

c.re=c1.re-re;

c.im=c1.im-im;

return c;

}

Complex mul(Complex c1)

{

Complex c;

c.re=(c1.re\*re)-(c1.im\*im);

c.im=(c1.re\*im)+(c1.im\*re);

return c;

}

};

int main()

{

Complex c1(2,3),c2(4,3.5),c;

c2=c.sumcom(c1);

c1.display();

c2.display();

c.display();

c=c1.negcom();

c.display();

c1.input();

cout<<"Subtraction: ";

c2=c.subcom(c1);

c2.display();

cout<<"Multiplication: ";

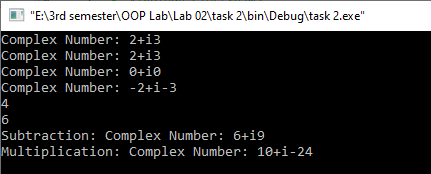
c2=c.mul(c1);

c2.display();

return 0;

}

**Output:**

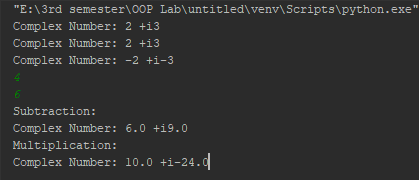


**In Python**

**Source code:**

class Complex:  
 def \_\_init\_\_(self):  
 self.re=0  
 self.im=0  
 def \_\_init\_\_(self,a,b):  
 self.re=a  
 self.im=b  
 def display(self):  
 print("Complex Number: "+str(self.re)+" +i"+str(self.im))  
 def sumcom(self,C1):  
 C=Complex(0,0)  
 C.re=self.re+C1.re  
 C.im=self.im+C1.im  
 return C  
 def negcom(self):  
 temp=Complex(0,0)  
 temp.re=-(self.re)  
 temp.im=-(self.im)  
 return temp  
 def input(self):  
 self.re=float(input())  
 self.im=float(input())  
 def subcom(self,C1):  
 C=Complex(0,0)  
 C.re=(C1.re)-(self.re)  
 C.im=(C1.im)-(self.im)  
 return C  
 def mul(self,C1):  
 C=Complex(0,0)  
 C.re=(C1.re\*self.re)-(C1.im\*self.im)  
 C.im=(C1.re\*self.im)+(C1.im\*self.re)  
 return C  
  
C1=Complex(2,3)  
C2=Complex(4,3.5)  
C=Complex(0,0)  
C2=C.sumcom(C1)  
C1.display()  
C2.display()  
C=C1.negcom()  
C.display()  
C1.input()  
print("Subtraction: ")  
C2=C.subcom(C1)  
C2.display()  
print("Multiplication: ")  
C2=C.mul(C1)  
C2.display()

**Output:**



**Conclusion:**

This program helps us in understanding the basic concepts of classes and objects in different languages. It also helps us understand **Returning from a function.**

# Activity # 03

**Title:**

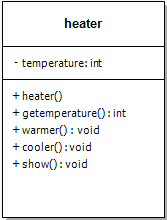
Make a Class for IntegerSet and model it using **Array**.

**Problem analysis:**

Create a class, **IntegerSet** that contains a single integer field, **Array[SIZE]**. Define a constructor that takes no parameters and sets all the elements of the array to 0. Define the function **UnionOfIntegerSet**, **IntersectionOfIntegerSet**, **NewIntegerSet,** **InsertElement, DeleteElement** and **IsEqualTo**. Define an accessor method to show the value of **Array**. Demonstrate the use of IntegerSet class.

**Algorithm:**

UML diagram for the above problem is given below:

* First make class IntegerSet

**IntegerSet**

* Declare Array as private integer field
* Define a no argument constructor.

-Array[SIZE]:int

* Define NewIntegerSet, InsertElement and DeleteElement.
* Define UnionOfIntegerSet and IntersectionOfIntegerSet.

+IntegerSet()

+NewIntegerSet(): void

+SetPrint(): void

+InsertElement(int): void

+DeleteElement(int):void

+UnionOfIntegerSet()

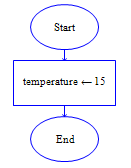
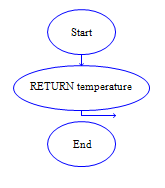
+IntersectionOfIntegerSet

* In main function, make object of IntegerSet to demonstrate theeuse of IntegerSet.
* Call each function one after the other and display the SetPrint function as shown in the flow chart.

**Flowchart:**

Start

Private: int Array[SIZE]

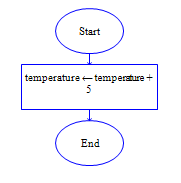


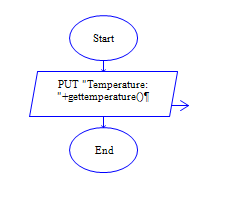
Array[i]=1;Array[j]=0

Array 🡨0

Void NewIntegerSet(int a)

IntegerSet()



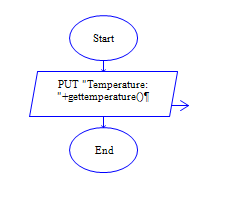


Array[a]=0

Array[a]=1

void InsertElement(int a)

void DeleteElement(int a)



If (Array[i]==1) cout<<i;

Void SetPrint()

End

**In C++**

**Source code:**

#include <iostream>

#include <cstdlib>

using namespace std;

const int SIZE=50;

class IntegerSet{

private:

int Array[SIZE];

public:

IntegerSet(){

for (int i=0;i<SIZE;i++)

{

Array[i]=0;

}

}

void NewIntegerSet(int A[])

{

for(int i=0;i<SIZE;i++)

{

for (int a=0;a<SIZE;a++)

{

if (A[a]==i)

Array[i]=1;

else

Array[i]=0;

}

}

}

void SetPrint()

{

int c=0;

for (int i=0;i<SIZE;i++)

{

if (Array[i]==1)

{

c++;

if (c>1)

cout<<", ";

cout<<i<<" ";

}

}

cout<<endl;

}

void InsertElement(int k)

{

Array[k]=1;

}

void DeleteElement(int m)

{

Array[m]=0;

}

IntegerSet UnionOfIntegerSet(IntegerSet A)

{

IntegerSet B;

for(int i=0;i<SIZE;i++)

{

if (A.Array[i] || Array[i])

B.Array[i]=1;

else

B.Array[i]=0;

}

return B;

}

IntegerSet IntersectionOfIntegerSet(IntegerSet A)

{

IntegerSet B;

for(int i=0;i<SIZE;i++)

{

if ((!(A.Array[i]) || !(Array[i])))

{

B.Array[i]=0;

}

else

B.Array[i]=1;

}

return B;

}

void IsEqualTo(IntegerSet A)

{

int counter=0;

for(int i=0;i<SIZE;i++)

{

if (A.Array[i]==Array[i])

counter++;

}

if (counter==SIZE)

cout<<"Integer Sets are Equal."<<endl;

else

cout<<"Integer Sets are Not Equal."<<endl;

}

};

int main()

{

int x[SIZE],y[SIZE];

for(int i=0;i<SIZE;i++)

{

x[i]=rand()%50;

}

for(int i=0;i<SIZE;i++)

{

y[i]=rand()%50;

}

IntegerSet i1,i2,i3;

i1.NewIntegerSet(x);

i1.SetPrint();

i2.NewIntegerSet(y);

i2.SetPrint();

i1.InsertElement(33);

i1.DeleteElement(44);

i3= i1.UnionOfIntegerSet(i2);

i3.SetPrint();

i3 = i1.IntersectionOfIntegerSet(i2);

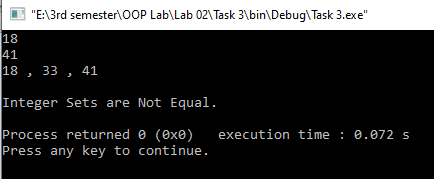
i3.SetPrint();

i1.IsEqualTo(i2);

return 0;

}

**Output:**



**In Python**

**Source code:**

SIZE=50  
import random  
  
class IntegerSet:  
  
 def \_\_init\_\_(self):  
 Array = []  
 self.Array=[0 for i in range(SIZE)]  
 def NewIntegerSet(self,A):  
 for i in range(SIZE):  
 c=0  
 for j in range(SIZE):  
 if i==A[j]:  
 c+=1  
 self.Array[i]=1  
 break  
 if c==0:  
 self.Array[i]=0  
 def SetPrint(self):  
 for i in range(SIZE):  
 if self.Array[i]==1:  
 print(i)  
 def InsertElement(self,k):  
 self.Array[k]=1  
 def DeleteElement(self,m):  
 self.Array[m]=0  
 def UnionOfIntegerSet(self,A):  
 B=IntegerSet()  
 for i in range(SIZE):  
 if (A.Array[i] or self.Array[i]):  
 B.Array[i]=1  
 else:  
 B.Array[i]=0  
 return B  
 def IntersectionOfIntegerSet(self,A):  
 B=IntegerSet()  
 for i in range(SIZE):  
 if (not A.Array[i] or not self.Array[i]):  
 B.Array[i]=0  
 else:  
 B.Array[i]=1  
 return B  
 def IsEqualTo(self,A):  
 c=0  
 for i in range(SIZE):  
 if A.Array[i]==self.Array[i]:  
 c+=1  
 if c==SIZE:  
 print("Integer Sets are Equal")  
 else:  
 print("Integer Sets are Not Equal")  
  
x=[random.randrange(0,50) for i in range(SIZE)]  
y=[random.randrange(0,50) for i in range(SIZE)]  
i1=IntegerSet()  
i2=IntegerSet()  
i3=IntegerSet()  
i1.NewIntegerSet(x)  
i1.SetPrint()  
i2.NewIntegerSet(y)  
i2.SetPrint()  
i1.InsertElement(33)  
i1.DeleteElement(44)  
i3=i1.UnionOfIntegerSet(i2)  
i3.SetPrint()  
i3=i1.IntersectionOfIntegerSet(i2)  
i3.SetPrint()  
i1.IsEqualTo(i2)

**Conclusion:**

This program helps us in understanding the basic concepts of classes and objects in different languages. It also helps us understand **Passing Arrays to Functions**.