
Lab No.2 Object Programming Essentials [Part 2]

2.1 Introduction

This lab has been designed to give in-depth knowledge about class objects and their usage as arguments and return type of class member function. It is also discussed how these member functions can be effectively tested using test cases.

2.2 Objectives of the lab

- 1 Understand how class object can be passed and returned from class member function
- 2 Write a class with member function having objects as arguments
- 3 Write a class with member function that return object
- 4 Test member function effectively using given test cases

2.3 Pre-Lab

2.3.1 Objects as function arguments

- 1 A member function has direct access to all the other members of the class – **public** or **private**
- 2 A function has indirect access to other objects of the same class that are passed as arguments
 - a Indirect access: using object name, a dot, and the member name

2.3.2 Example code: complexObjfun.cpp

It must be noted that in C++, member functions defined inside a class are *inline* by default. Declaration and definition can be separated from each other as shown in example code. Its format and syntax is as follows:

- Declaration is written inside the class
- Definition is written outside the class

Syntax:

```
Return_type class_name :: ftn_name (list of parameters){  
    // Body of function  
}
```

where :: operator is called **scope resolution operator**. It specifies what class something is associated with.

```
#include <iostream>  
using namespace std;  
class complex{
```

```

private:
    double re, im;
public:
    complex();
    complex(double r, double i);
    void addCom(complex c1, complex c2);    // Note: add() takes objects as an arguments
    void show();
};

complex::complex(): re(0), im(0){}    // Note: style of constructor here

complex::complex(double r, double i): re(r), im(i){} // Note: style of overloaded constructor here

void complex::addCom (complex c1, complex c2){
    re=c1.re + c2.re;
    im=c1.im + c2.im;
}

void complex::show(){
    cout<<re<<"+"<<im<<"i"<<endl;
}

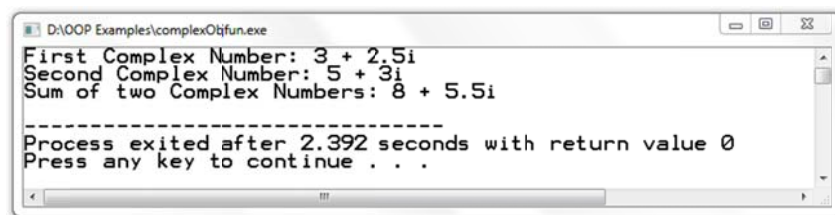
int main(){
    complex c1(3,2.5),c2(5,3),c;
    c.addCom (c1, c2);    // Note: object c1 and c2 are passed

    cout << "First Complex Number: "; c1.display();
    cout << "Second Complex Number: "; c2.display();
    cout << "Sum of two Complex Numbers: "; c.display();

    return 0;
}

```

Output:



```

D:\OOP Examples\complexObjfun.exe
First Complex Number: 3 + 2.5i
Second Complex Number: 5 + 3i
Sum of two Complex Numbers: 8 + 5.5i

-----
Process exited after 2.392 seconds with return value 0
Press any key to continue . . .

```

Figure 2.1: Output complexObjfun.cpp

2.3.3 Returning objects from functions

Objects can be returned from functions just like normal primitive type variables.

2.3.4 Example code in C++: complexRetObjfun.cpp

```
#include <iostream>
using namespace std;
class complex {
private:
    double re, im;
public:
    complex();
    complex(double r, double i);
    complex negate();           // Note: negate returns complex object
    void show();
};
complex::complex(): re(0), im(0){}

complex::complex(double r, double i): re(r), im(i){}

complex complex::negate() {
    complex temp;
    temp.re= - re;
    temp.im= - im;
    return temp;
}

void complex::show() {
    if(im>0)
        cout << re << " + " << im << "i" << endl;
    else
        cout << re << im << "i" << endl;
}

int main() {
    complex c1(3,2.5),c2;
    c2=c1.negate();           // Note: negate() returning object in c2

    cout << "Complex Number: "; c1.display();
    cout << "Negation of Complex Number: "; c2.display();

    return 0;
}
```

Output:

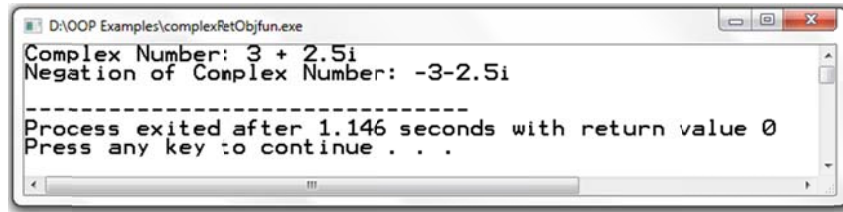


Figure 2.2: Output complexRetObjfun.cpp

2.3.5 Example code in Java: complexObjfun.java

```
package lab2s1;

public class complexObjfun {
    private double re,im;

    public complexObjfun() {
        this.re=0;
        this.im=0;
    }

    public complexObjfun(double r, double i) {
        this.re=r;
        this.im=i;
    }

    public void addCom(complexObjfun c1, complexObjfun c2) {
        this.re = c1.re + c2.re;
        this.im = c1.im + c2.im;
    }

    public complexObjfun negate() {
        complexObjfun temp = new complexObjfun();
        temp.re = -1.0*this.re;
        temp.im = -1.0*this.im;
        return temp;
    }

    public void show() {
        if (im>0)
            System.out.println(this.re+" "+this.im+"i");
        else {
            System.out.print(this.re);
```

```

        System.out.println(this.im+"i");
    }
}
}

```

Test.java

```

package lab2s1;

public class Test {
    public static void main(String[] args) {
        complexObjfun c1 = new complexObjfun(3,2.5);
        complexObjfun c2 = new complexObjfun(5,3);
        complexObjfun c = new complexObjfun();

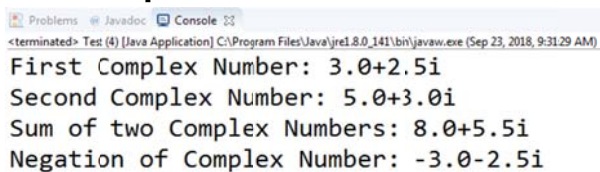
        c.addCom(c1, c2);

        System.out.print("First Complex Number: ");
        c1.show();
        System.out.print("Second Complex Number: ");
        c2.show();
        System.out.print("Sum of two Complex Numbers: ");
        c.show ();

        c=c1.negate();
        System.out.print("Negation of Complex Number: ");
        c.show ();
    }
}

```

Output:



```

<terminated> Test (4) [Java Application] C:\Program Files\Java\jre1.8.0_141\bin\javaw.exe (Sep 23, 2018, 9:31:29 AM)
First Complex Number: 3.0+2.5i
Second Complex Number: 5.0+3.0i
Sum of two Complex Numbers: 8.0+5.5i
Negation of Complex Number: -3.0-2.5i

```

Figure 2.3: Output complexObjfun.java & Test.java

2.3.6 Example code in Python: complexObjfun.py

```

class complex:
    def __init__(self):
        self.re = 0.0
        self.img = 0.0

```

```

def __init__(self,r,i):
    self.re = r
    self.img = i
def addCom(self,num1,num2):
    self.re=num1.re+num2.re
    self.img=num1.img+num2.img
def negate(self):
    self.re=-self.re
    self.img=-self.img
    return self
def show(self):
    if self.img>0:
        print self.re,"+",self.img,"i"
    else:
        print self.re,self.img,"i"

c1=complex(3,2.5)
print "First Complex Number: "
c1.show()

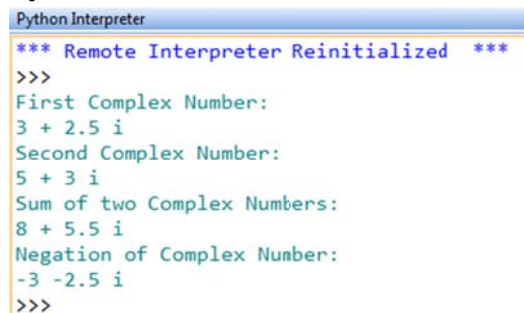
c2=complex(5,3)
print "Second Complex Number: "
c2.show()

c=complex(0,0)
print "Sum of two Complex Numbers: "
c.addCom(c1,c2)
c.show()

c=c1.negate()
print "Negation of Complex Number: "
c.show()

```

Output:



```

Python Interpreter
*** Remote Interpreter Reinitialized ***
>>>
First Complex Number:
3 + 2.5 i
Second Complex Number:
5 + 3 i
Sum of two Complex Numbers:
8 + 5.5 i
Negation of Complex Number:
-3 -2.5 i
>>>

```

Figure 2.4: Output complexObjfun.py

2.4 Activities

Perform all these activities in C++, Java and Python.

2.4.1 Activity

Reuse **Complex** class given in **section 2.3.2 (C++)**, **section 2.3.5 (Java)**, and **section 2.3.6 (Python)** to perform arithmetic with complex numbers. Note that addition and printing is already done in given sections. Add the following public methods to perform complex subtraction and multiplication as well:

a) **Input:** Write class function input() to take complex number real and imaginary parts from user on runtime. Note: input takes no arguments and returns nothing.

b) **Subtract two Complex numbers:** Write class function subCom() taking two complex objects c1 and c2. Difference is computed as following: the real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand. Note: return type of subCom is void.

c) **Multiply two Complex numbers:** Write class function mulCom() taking two complex objects c1 and c2. Product is computed as following: Suppose you are trying to compute the product of two complex numbers $a + bi$ and $c + di$. The real part will be $ac - bd$, while the imaginary part will be $ad + bc$. Note: return type of subCom is void.

Demonstrate and test the modified class and its objects using test cases given in Section 2.5.

2.4.2 Activity

Reuse **Complex** class written in **Activity 2.4.1** to modify the addCom(), subCom(), and mulCom() class functions. Instead of passing two objects in each, now pass only one object. Change the return type of each function to complex. Adjust the function code to match the changes. Demonstrate and test the modified class and its objects using test cases given in Section 2.5.

2.4.3 Activity

Create a class called **IntegerSet**. Each object of class **IntegerSet** can hold integers in the range 0 through 49. A set is represented internally as an array of ones and zeros. Array element $a[i]$ is 1 if integer i is in the set. Array element $a[j]$ is 0 if integer j is not in the set. The default constructor initializes a set to empty set i.e., a set whose array representation contains all zeros.

Provide member functions for the common set operations. For example,

1. Provide a **newIntegerSet** member function that initialize array to user-defined array provided as an input in array notation. Note function return type is void.
2. Provide a **unionOfIntegerSets** member function that creates a third set which is the set-theoretic union of two existing sets (i.e., an element of the third set's array is set to 1 if that element is 1 in either or both of the existing sets, and an element of the third set's array is set to 0 if that element is 0 in each of the existing sets).
3. Provide an **intersectionOfIntegerSets** member function that creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to 0 if that element is 0 in either or both of the existing sets, and an element of the third set's array is set to 1 if that element is 1 in each of the existing sets).
4. Provide an **insertElement** member function that inserts a new integer k into a set (by setting a[k] to 1).
5. Provide a **deleteElement** member function that deletes integer m (by setting a[m] to 0).
6. Provide a **setPrint** member function that prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set (i.e., their position in the array has a value of 1).
7. Provide an **isEqualTo** member function that determines if two sets are equal.

Now write a driver program to test your **IntegerSet** class. Instantiate several **IntegerSet** objects. Test that all your member functions work properly.

2.5 Testing

Test Cases for Activity 2.4.1 and 2.4.2

Sample Inputs	Sample Outputs
Declare three Complex Objects c1, c2, and c Take c1 real and imaginary values by calling input() Display content of c1 using show() Take c2 real and imaginary values by calling input() Display content of c2 using show() Call addCom() with c1 and c2 as input and c as output Display content of c using show() Call subCom() with c1 and c2 as input and c as output Display content of c using show() Call mulCom() with c1 and c2 as input and c as output Display content of c using show()	c1 = 3.0 + 2.5i c2 = 5.0 + 3.0i Sum = 8.0 + 5.5i Difference = -2.0 - 0.5i Product = 7.5 + 21.5i

Test Cases for Activity 2.4.3

Sample Inputs	Sample Outputs
Declare two integer arrays x and y of size 50 Initialize arrays randomly between 1 and 50 Declare three IntegerSet Objects i1,i2, and i3 Initialize i1 with x using NewIntegerSet() and display using setPrint() Initialize i2 with y using NewIntegerSet() and display using setPrint() Take location from user to insertElement() in i1 Take location from user to deleteElement() in i1 Call unionOfIntegerSets() with i1 and i2 as input and i3 as output Display content of i3 using setPrint() Call intersectionOfIntegerSets() with i1 and i2 as input and i3 as output Display content of i3 using setPrint() Check if the i1 and i2 are equal using isEqualTo()	Display First Set values Display Second Set values 33 44 Display Union Set values Display Intersection Set values Equal or not equal message

2.6 References:

- 1 Class notes
- 2 Object-Oriented Programming in C++ by *Robert Lafore* (Chapter 6)
- 3 How to Program C++ by *Deitel & Deitel* (Chapter 6)
- 4 Programming and Problem Solving with Java by *Nell Dale & Chip Weems*
- 5 Murach's Python Programming by *Micheal Urban & Joel Murach*