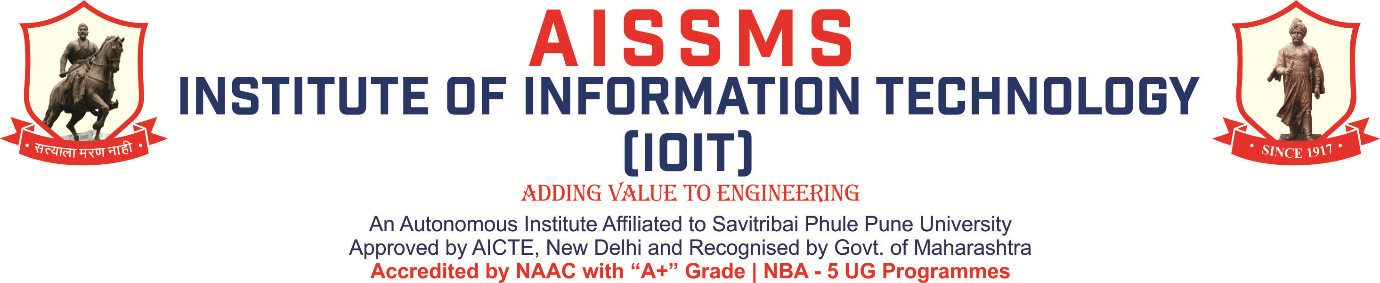
****

**All India Shri Shivaji Memorial Society’s**

**Institute of Information Technology**

**Pune**

**Department of Computer Engineering**

**LABORATORY MANUAL**

**Object Oriented Programming Laboratory**

SY BTech Computer Engineering

SEMESTER-I

## Subject Code: COPCC308

**TEACHING SCHEME EXAMINATION SCHEME**

Practical: 2 Hrs/Week Practical:50 Marks

**OOP Index**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Assi. No. | Title of Assignment | **Page**  **No** | **Date of**  **Conduction** | | **Date of**  **Submission** | | **Sign** |
| Group A | | | | | | |  |
|  | Write a C++ program to Generate series of prime numbers. |  | |  | |  |  |
|  | Write a C++ program implement simple Arithmetic Calculator |  | |  | |  |  |
|  | Write a C++ program to create class with usage of all access specifiers. |  | |  | |  |  |
|  | Write a CPP to create class Student with appropriate member variable and member functions and make use of following   1. Constructors 2. Destructors 3. Inline, static, friend function |  | |  | |  |  |
|  | Write a CPP to implement following inheritances using car rental system   1. Single  Inheritance 2. Multilevel inheritance 3. Multiple Inheritance 4. Hierarchical Inheritance |  | |  | |  |  |
| Group B | | | | | | |  |
|  | Write a CPP to implement Online Payment system using function overloading for Online Shopee. |  |  | |  | |  |
|  | Implement a class Complex which represents the Complex Number data type. Implement the following operations:  a. Constructor (including a default constructor which creates the complex number 0+0i).  b. Overloaded operator +, - to add and subtract two complex numbers  c. Overloaded operator \* , /to multiply and divide two complex numbers.  d. Overloaded << and >> to print and read Complex Numbers |  |  | |  | |  |
|  | Implement CPP to demonstrate Exception Handling for Gmail Account Login OR ATM Pin Verification. |  |  | |  | |  |
|  | Write a C++ program to implement stack of characters and integers using function template. |  |  | |  | |  |
|  | Implement Virtual function: Create a base class Employee with a virtual function calculateSalary() returning a base salary (e.g., 1000). Derive two classes: Manager, overriding calculateSalary() to add a bonus (e.g., 500), and Developer, overriding it to add a project bonus (e.g., 200). Implement a function that accepts an Employee\* and prints the salary by calling calculateSalary(). Finally, create instances of Manager and Developer, and pass them to the function to demonstrate polymorphism, ensuring the correct salary is calculated based on the actual object type. |  |  | |  | |  |
|  | Implement a function template that finds the maximum of two values of any data type (e.g., integers, floats, strings).  Create a class template Stack that implements a basic stack data structure. The class should support push, pop, as well as checking if the stack is empty. |  |  | |  | |  |
| Group C | | | | | | |  |
|  | Implement following functionality using file handling.   1. Create and write student object. 2. Display student details. 3. Search a record based on Roll no. and name 4. Modify a student record 5. Delete student record |  |  | |  | |  |
|  | Write a C++ program to generate Country-Currency chart of all countries across the globe using MAP Container |  |  | |  | |  |
|  | Write a C++ program  using Vector -Create a vector of integers, Add at least 5 elements to the vector, Display the contents of the vector, Access and print the 3rd element using both index and at() method. Remove the last element from the vector and print the updated vector |  |  | |  | |  |
|  | Write a C++ program using List- Create a list of integers and initialize it with the following values: 10, 20, 30, 40, 50.Perform the following operations :Add an element with the value 60 at the front of the list. Add an element with the value 5 at the back of the list. Remove the first and last element from the list. Print the contents of the list after each operation. Implement a function to display all elements of the list. |  |  | |  | |  |

**Group A**

**Assignment – 1**

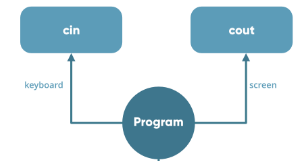
Title: Write a C++ program implement simple Arithmetic Calculator with Class and Without Class.

Objective: To understand basic C++ program for user input and display output

Theory:

C++ comes with libraries that provide us with many ways for performing input and output. In C++ input and output are performed in the form of a sequence of bytes or more commonly known as **streams**.

* **Input Stream:** If the direction of flow of bytes is from the device (for example, Keyboard) to the main memory then this process is called input.
* **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device (display screen ) then this process is called output.



**Basic concepts of Object-oriented programming:**

Object: Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

class:The entire set of data and code of an object can be made of a user defined data type with the help of a class in fact Objects are variables of the type class. Once a class has been defined , we can create any number of objects belonging to that class. A class is thus a collection of objects of similar type.

For example: mango, apple, and orange are members of the class fruit. ex: fruit mango; will create an object mango belonging to the class fruit. Data

Abstraction and Encapsulation: the wrapping up of data and functions in to a single unit is known as encapsulation.

Data encapsulation is the most striking feature of a class. The data is not accessable to the outside world and only those functions which are wrapped in the class can access. This insulation of the data from direct access by the program is called data hiding.

Abstraction : Abstraction refers to the act of representing essential features without including the background details or explanations. since the classes use the concept of data abstraction ,thy are known as abstraction data type(ADT).

Inheritance : Interitance is the process by which objects of one class acquire the properities of objects of another class. for example: The bird 'robin ' is a part of the class 'flying bird' which is again a part of the class 'bird'. The concept of inheritance provide the idea of reusability. POLYMORPHISM: ploymorphism is another important oop concept. Ploymorphism means the ability to take more than one form.an operation may exhibit different instances. The process of making an operator to exhibit different behaviours in different instance is known as operator overloading. Input/output Statements:: Cout<>n; Class definition: class definition has two components: the class head, the class body. Class vector //the class head { // all class members };

Algorithm:  
1. Start  
2. Take two numbers and an operator  
3. Use switch-case to perform +, -, \*, /  
4. Display result  
5. End

Sample Program

#include<iostream>

int main()

{

int a,b,c;

cin>>a>>b; // user input

c= a+b; //addition

cout<<c; //display output on screen

return 0;

}

**Simple calculator using Switch**

# include <iostream>

using namespace std;

int main() {

char op;

float num1, num2;

cout << "Enter operator: +, -, \*, /: ";

cin >> op;

cout << "Enter two operands: ";

cin >> num1 >> num2;

switch(op) {

case '+':

cout << num1 << " + " << num2 << " = " << num1 + num2;

break;

case '-':

cout << num1 << " - " << num2 << " = " << num1 - num2;

break;

case '\*':

cout << num1 << " \* " << num2 << " = " << num1 \* num2;

break;

case '/':

cout << num1 << " / " << num2 << " = " << num1 / num2;

break;

default:

// If the operator is other than +, -, \* or /, error message is shown

cout << "Error! operator is not correct";

break;

}

return 0;

}

**Calculator Using Class**

#include<iostream>

char op;

using namespace std;

class calculator

{

float num1, num2, result;

public:

void getData();

void putData();

void add();

void sub();

void mul();

void div();

};

void calculator::getData()

{

cout<<"Enter num1, operator and num2\t";

cin>>num1>>op>>num2;

}

void calculator::putData()

{

cout<<"Num1 = "<<num1<<"\tNum2 = "<<num2<<"\tResult = " <<result;

}

void calculator::add()

{

result=num1+num2;

}

void calculator::sub()

{

result=num1-num2;

}

void calculator::mul()

{

result=num1\*num2;

}

void calculator::div()

{

result=num1/num2;

}

int main()

{

char ans;

calculator c;

do

{

c.getData();

switch(op)

{

case '+':

c.add();

break;

case '-':

c.sub();

break;

case '\*':

c.mul();

break;

case '/':

c.div();

break;

}

c.putData();

cout<<"\nDo you want to continue(Y/N)";

cin>>ans;

}while(ans=='Y' || ans=='y');

return 0;

}

**Conclusion**: Implemented the simple calculator without class and using class.

**Assignment – 2**

## Title: Write a CPP to create class Student with appropriate member variable and member functions and make use of following

1. Constructors
2. Destructors
3. Inline, static, friend function

**Objective**: Understand C++ Class, Constructors, Destructors, Inline, static, friend function and Dynamic memory allocation-deallocation

**Theory**:

A special method of the class that will be automatically invoked when an instance of the class is created is called as constructor. Following are the most useful features of constructor.

1. Constructor is used for Initializing the values to the data members of the Class.
2. Constructor is that whose name is same as name of class.
3. Constructor gets Automatically called when an object of class is created.
4. Constructors never have a Return Type even void.
5. Constructor is of Default, Parameterized and Copy Constructors

**Types of constructors**

There are three types of constructors -

* **Default constructor -** A default constructor is one that does not have function parameters. It is used to initialize data members with a value. The default constructor is called when the object is created.
* **Parameterized constructor -** A non-parameterized constructor does have the constructor arguments and the value passed in the argument is initialized to its data members. Parameterized constructors are used in constructor overloading.
* **Copy constructor -** Copy constructor initializes an object using another object of the same class.
  + **Syntax class\_name(constclassname&old\_object).**

// C++ program to demonstrate constructor overloading

#include <iostream>

using namespace std;

class Person {

private:

int age;

public:

// 1. Constructor with no arguments

Person() {

age = 20;

}

// 2. Constructor with an argument

Person(int a) {

age = a;

}

int getAge() {

return age;

}

};

int main() {

Person person1, person2(45);

cout << "Person1 Age = " << person1.getAge() << endl;

cout << "Person2 Age = " << person2.getAge() << endl;

return 0;

}

**Destructor:** As we know that Constructor is that which is used for Assigning Some Values to data Members and for Assigning Some Values this May also used Some Memory so that to free up the Memory which is Allocated by Constructor, destructor is used which gets Automatically Called at the End of Program and we doesn’t have to Explicitly Call a Destructor and Destructor Cant be Parameterized or a Copy This can be only one Means Default Destructor which Have no Arguments. For Declaring a destructor we have to use ~tiled Symbol in front of Destructor.

**Static members:** A class can contain *static* members, either data or functions. A static member variable has following properties:

* + - It is initialized to zero when the first object of its class is created. No other initialization is

permitted.

* + - Only one copy of that member is created for the entire class and is shared by all the objects of that class.
    - It is the visible only within the class but its lifetime is the entire program.

Static data members of a class are also known as "class variables", because there is only one unique value for all the objects of that same class. Their content is not different from one object static members have the same properties,

A static member function has following properties :

1. A static function can have access to only other static members (fun or var) declared in the same class
2. A static function can be called using the class name instead of its object name

*Class\_name :: function\_name;*

Static member functions are considered to have class scope. In contrast to non static member functions, these functions have no implicit **this** argument; therefore, they can use only static data members, enumerators, or nested types directly. Static member functions can be accessed without using an object of the corresponding class type.

The following restrictions apply to such static functions:

* + - They cannot access non static class member data using the member-selection operators (**.** or **–>**).
    - They cannot be declared as **virtual**.
    - They cannot have the same name as a non static function that has the same argument types.

Ex. Shall we give the example………………………

### Friend functions:

In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect *friends*. Friends are functions or classes declared as such. If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword *friend.*

### Properties of friend function:

* + - It is not in the scope of the class to which it has been declared as friend.
    - Since it is not in the scope of the class , it cannot be called using the object of that class
    - It can be invoked like a normal function w/o the help of any object.
    - It can be declared in private or in the public part of the class.
    - Unlike member functions, it cannot access the member names directly and has to use an object name and dot operator with each member name.

### Friend classes

Just as we have the possibility to define a friend function, we can also define a class as friend of another one, granting that second class access to the protected and private members of the first one.

**Inline Function**

C++ provides inline functions to reduce the function call overhead. An inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call. This substitution is performed by the C++ compiler at compile time. An inline function may increase efficiency if it is small.

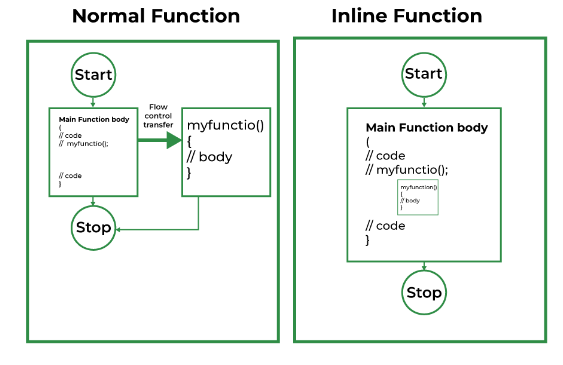
Syntax to declare inline function is as follows:

inline return-type function-name(parameters)

{

// function code

}



**Conclusion**: C++ student database program is implemented using constructor, destructor by making use of dynamic memory allocation and use of friend and inline function.

**Assignment 3**

|  |
| --- |
| **Title**: Write a CPP to implement following inheritances using car rental system.  a. Single Inheritance  b. Multilevel inheritance  c. Multiple Inheritance  d. Hierarchical Inheritance  **Objective**: To Study inheritance concept in C++ |

**Theory**:

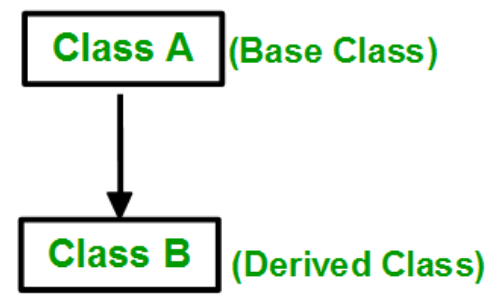
he capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.

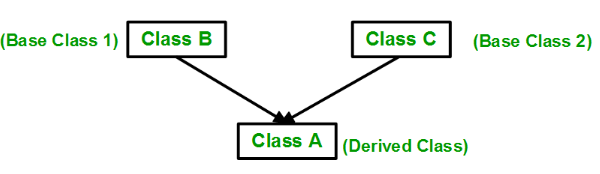
**Types Of Inheritance:-**

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

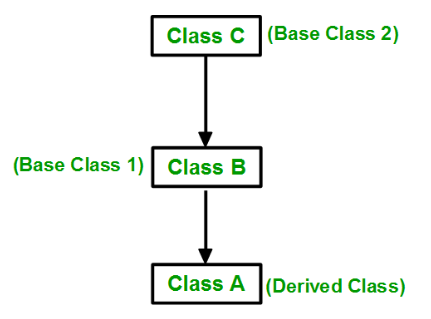
**Single Inheritance**: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.



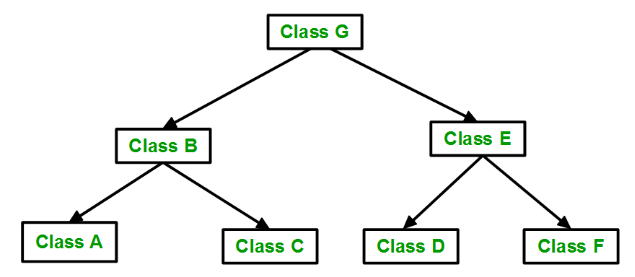
**Multiple Inheritance**: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one subclass is inherited from more than one base class.



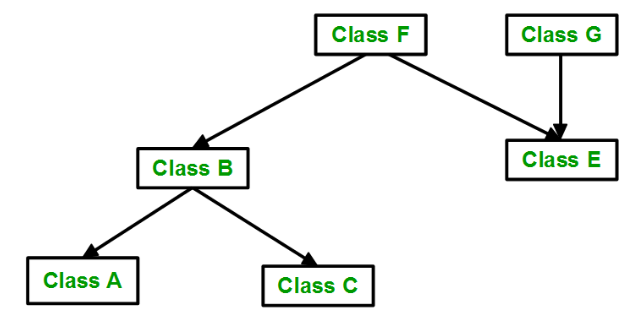
**Multilevel Inheritance**: In this type of inheritance, a derived class is created from another derived class.



**Hierarchical Inheritance**: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



**Hybrid (Virtual) Inheritance**: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance. Below image shows the combination of hierarchical and multiple inheritances:



**Conclusion:** Car rental system is implemented using the concept of inheritance

**Assignment-4**

**Title**: Write a CPP to implement Online Payment system using function overloading for Online Shoppe.

**Objective**: Learn syntax and semantic of function overloading

**Theory:** Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading “Function” name should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++.

With function overloading, multiple functions can have the same name with different parameters:

**Example**

int myFunction(int x);

float myFunction(float x);

double myFunction(double x, double y);

The parameters should follow any one or more than one of the following conditions for Function overloading:

Parameters should have a different type

add(int a, int b)

add(double a, double b)

**Sample Program**

#include<iostream>

using namespace std;

void add(int a, double b)

{

cout<<"sum = "<<(a+b);

}

void add(double a, int b)

{

cout<<endl<<"sum = "<<(a+b);

}

// Driver code

int main()

{

add(10,2.5);

add(5.5,6);

return 0;

}

**Conclusion**: Online payment System is implemented using Function overloading

**Assignment-4**

**Title** : Implement a class Complex which represents the Complex Number data type. Implement the following operations:

a. Constructor (including a default constructor which creates the complex number 0+0i).

b. Overloaded operator +, - to add and subtract two complex numbers

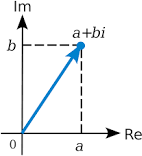
c. Overloaded operator \*, /to multiply and divide two complex numbers.

d. Overloaded << and >> to print and read Complex Numbers.

**Objective**: Implement arithmetic operation in Complex number using the concept of Operator Overloading

**Theory:**

A **complex number** is a **number** that can be expressed in the form a + bi, where a and b are real **numbers** and i is the imaginary unit, that satisfies the equation i2 = −1. In this expression, a is the real part and b is the imaginary part of the **complex number**.



### Operations on Complex number:

1. **Addition of Complex Numbers** Addition of two complex numbers a + b i and c + d i is

defined asfollows.

(a + b i) + (c + d i) = (a + c) + (b + d) i

This is similar to grouping like terms: real parts are added to real parts and imaginary parts are added to imaginary parts.

### Subtraction of Complex Numbers

The subtraction of two complex numbers a + b i and c + d i is defined as follows. (a + b i) - (c + d i) = (a - b) + (b - d) i

### Mulyiply Complex Numbers

The multiplication of two complex numbers a + b i and c + d i is defined as follows. (a + b i)(c + d i) = (a c - b d) + (a d + bc) i

However you do not need to memorize the above definition as the multiplication can be carried out using properties similar to those of the real numbers and the added property i 2 = -1. (see the examplebelow)

### OOP Features & Concepts used in this practical :

**Operator Overloading:** You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Box operator+(const Box&);

declares the addition operator that can be used to **add** two Box objects and returns final Box object.

### Overloadable/Non-overloadableOperators:

Following is the list of operators which can be overloaded:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| + | | - | | \* | | / | | % | | ^ | |
| & | | | | | ~ | | ! | | , | | = | |
| < | | > | | <= | | >= | | ++ | | -- | |
| << | | >> | | == | | != | | && | | || | |
| += | | -= | | /= | | %= | | ^= | | &= | |
| |= | | \*= | | <<= | | >>= | | [ ] | | ( ) | |
| -> | | ->\* | | new | | new [ ] | | delete | | delete [ ] | |

**Operator that are not overloaded** are follows

scope operator - :: sizeof

member selector - . member pointer selector - \* ternary operator - ?:

In C++, Operator overloading is a compile-time polymorphism. It is a method of giving special meaning to an existing operator in C++ without changing its original meaning. C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. Operator overloading is a compile-time polymorphism. For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc.

Syntax:

return type operator opr(list of parameters)

class className {

... .. ...

public

returnType operator symbol (arguments) {

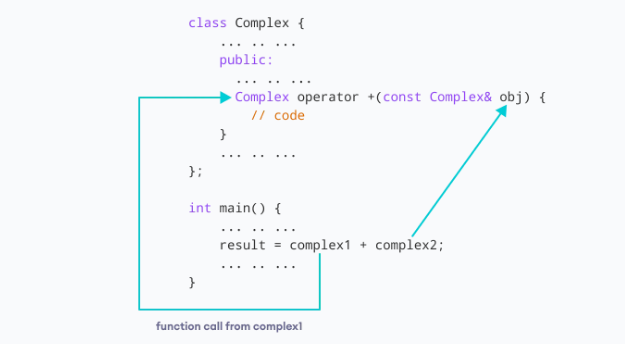
... .. ...

}

... .. ...

};

Overloading Binary operators



Operator (Unary Operator) Overloading

// Overload ++ when used as prefix and postfix

#include <iostream>

using namespace std;

class Count {

private:

int value;

public:

// Constructor to initialize count to 5

Count() : value(5) {}

// Overload ++ when used as prefix

void operator ++ () {

++value;

}

// Overload ++ when used as postfix

void operator ++ (int) {

value++;

}

void display() {

cout << "Count: " << value << endl;

}

};

int main() {

Count count1;

// Call the "void operator ++ (int)" function

count1++;

count1.display();

// Call the "void operator ++ ()" function

++count1;

count1.display();

return 0;

}

Sample Code:

#include <iostream>          
using namespace std;  
class complex               //class name complex is declared  
{  
    float realp,imagp;          
public:  
    complex()                   //default constructor  
    {  
        realp=0;  
        imagp=0;  
    }  
    complex operator+(complex &);       //for addition of two complex nos  
    complex operator\*(complex &);       //for multiplication of two complex nos  
    complex(float,float);               //parameterized constructor  
    friend istream &operator>>(istream &,complex &);  
    friend ostream &operator<<(ostream &,complex &);  
};  
complex::complex(float x,float y)       //parameterized constructor definition  
{  
    realp=x;  
    imagp=y;  
}  
//function to accept values of real and imag parts of complex no  
istream &operator>>(istream &din,complex &c)     
{  
    cout<<"Enter real part of complex number 2: ";  
    din>>c.realp;  
    cout<<"\nEnter imaginary part of complex number 2: ";  
    din>>c.imagp;  
    return din;  
}  
//functions to display complex nos  
ostream &operator<<(ostream &dout , complex &c)  
{  
    dout<<c.realp<<" + "<<c.imagp<<"i";  
    dout<<endl;  
    return dout;  
}  
//function to add two complex nos  
complex complex::operator+(complex &c)  
{  
    complex temp;  
    temp.realp=realp + c.realp;   
    temp.imagp=imagp + c.imagp;   
    return temp;  
}  
//function to multiply two complex nos  
complex complex::operator\*(complex &c)  
{  
    complex mul;  
    mul.realp=(realp\*c.realp) - ( imagp\*c.imagp);  
    mul.imagp=(imagp\*c.realp) + (realp\*c.imagp);  
    return mul;  
}  
int main()  
{  
    complex c2, c3;       
    complex c1(1.2,2.2);     
    cout<<"Complex no 1 is:"<<c1;  
    cout<<"Enter complex no 2:\n";  
    cin>>c2;        
    cout<<"Complex number 1 is :";  
    cout<<c1;  
    cout<<"Complex number 2 is :";  
    cout<<c2;  
    cout<<"Complex number 3 is :";  
    cout<<c3;  
    cout<<"\nAddition of two complex numbers is: ";  
    c3=c1+c2;  
    cout<<c3;  
    cout<<"\nMultiplication of two complex number is: ";  
    c3=c1\*c2;        
    cout<<c3;       //display value of c3  
    return 0;  
}

**Conclusion**: Complex number arithmetic is implemented using Operator overloading