ASSIGNMENT 01

Q:1 What is Object Oriented Programming ? Explain the features of Object Oriented Programming (OOP).

Ans: Object Oriented Programming - as the name suggests uses objects in programming.
Object Oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming
The main aim of OOP is to bind together the data and the functions that operates on them so that no other part of the code can access this data except that function.

* Features of OOP are:

I) **CLASS** : Class is a collection of data members and member functions. It is also known as group of objects
Class is a structure or a body where we can define data members and member functions to be utilised by objects.

II) **OBJECT** : Objects are the instances of class. Objects are the basic runtime entities in the OOP. they may represent a person, place etc. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

III) **DATA ABSTRACTION & HIDING** : Abstraction refers to the act of representing essential features without including background details. Basically it hides all the implementation details.

IV) **ENCAPSULATION** : The wrapping of data & functions into a single unit is known as encapsulation.
Data Encapsulation is the most important feature of a class.

v) **INHERITANCE** : Inheritance is the process by which objects of one class aquire the properties of another class object. when one class is derived from another class its known as Inheritance. It has 5 types :

a) Single level
b) Multi level
c) Multiple level
d) Hybrid level
e) Hierarchical level

vi) **POLYMORPHISM** : It is a feature of OOP which create function with the same name but with different arguments which will perform different different tasks.

Types of polymorphism :

a) RunTime (DYNAMIC)
b) CompileTime (STATIC)

vii) **DYNAMIC BINDING** : It is perjormed on Run-Time and refers to the linking of a procedural call (function) to the code (function definition) to be executed in response to the call. It is also known as Late Binding.

viii) **MESSAGE PASSING** : In this, Objects communicate with each other via sending or recieving the information at runtime.

Q:2 Why is C++ better than C ? Also explain the C++ variables and C++ Scope Resolution operator by the example of a Program.

Ans: Despite being truly object-Oriented, supporting procedural programming makes C++ much like a hybrid programming language. Being object-oriented means that C++ enhances productivity as well as the organisation of the code. This is because C++ has higher level of abstraction than C programming language and is better than C.

* **C++ VARIABLES :** Variables are used in C++ to store any type of values within a program and whose value can be changed during the program execution.

Syntax :

```
data_type variable_name ;
```

* **SCOPE RESOLUTION OPERATOR :** This operator in C++ programming language is used to define a function outside a class or when we want to use a global variable but also has a local variable with the same name. (::)

Syntax :

```
return_type class_name :: function_name ()
{                                       }
```

* **EXAMPLE :**

```
class student
{  int roll_no ;                          → DATA MEMBER WITH VARIABLE NAME
   public : void readshow () ;            → MEMBER FUNCTION
};
void student :: readshow ()               → USE OF SCOPE RESOLUTION OPERATOR
{ cout << "Enter Rollno : " ;
  cin >> roll_no ;                        } DEFINITION OF READSHOW FUNCTION
  cout << roll_no ;
}
void main ()
{ student st1 ;                           → VARIABLE DECLARATION with student DATATYPE.
  st1 . readshow () ;
  getch () ;
}
```

Q:3 Explain Data Abstraction and Data Encapsulation.

Ans: **DATA ABSTRACTION :**

Abstraction refers to the act of representing essential features without including background detail / explanation (Basically it hides all the implementation details). When a class is given to the user, he is familiar only with the public interface of the class.
Internal details and implementation are hidden from the user. Example : PRINTF, SCANF function.

DATA ENCAPSULATION :

The wrapping of data and functions into a single unit (class) is known as Encapsulation.
Data Encapsulation is the most important feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

Q:4 Explain Inheritance and Polymorphism.

Ans: **INHERITANCE :**

It is a process by which object of one class aquire the properties of another class object. When one class is derived from another class it is known as Inheritance.
Types of Inheritance :

a) single level Inheritance :
When one class is derived from another class is known as a single level inheritance.

b) Multi level Inheritance :
If a class is derived from a class and further class derived class. This is known as Multi level Inheritance.

c) Hierarchical level Inheritance :
If 2 or more classes are derived from a single class (BASE) is known as Hierarchical level Inheritance.

d) Multiple Level Inheritance :
when a class is derived from more than one base class then it is called as Multiple Level Inheritance.

e) Hybrid Level Inheritance :
Combination of any 2 level of inheritance is known as Hybrid level Inheritance.

POLYMORPHISM :

It is a feature of object oriented Programming which create function with the same name but with different arguments which will perform different different tasks.
(One name, Many forms).

Example :

```
int sum (int x, int y)
{          }
void main ()
{ x * y ; }
```

```
float sum (float x, float y)
{          }
void main ()
{ int *p ; }
```

Q:5 Explain the PROCEDURAL, STRUCTURED and OOPs PARADIGM in detail with examples.

Ans: **PROCEDURAL PARADIGM :**

Procedural Programming is a programming paradigm, derived from structured programming, based on the concept of the procedure call.
Procedures, also known as routines, subroutins or functions, simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution.
Any given procedure might be called at any point during a program's execution, including by other procedures or itself.

* **STRUCTURED PARADIGM:**

structured programming is a programming paradigm aimed at improving the clarity, quality and development time of a computer program by making extensive use of structured control flow constructs of selection (if / then / else) and repetition (while and for), block structures and sub-routines.

* **OBJECT ORIENTED PROGRAMMING PARADIGM:**

The major motivating factor in the invention of OO approach is to remove some of the flaws encountered in the procedural approach.

OOP treats data as a critical element in the program development and does not allow it to flow freely around the systems.

The data of an object can be accessed only by the function associated with that object. However, functions of one object can access the functions of other objects.

ASSIGNMENT 02

Q:1 How we can define a function inline, what are the benefits of this methodology ?

Ans: Inline functions are the functions whose difinition ends in one line only.

Defining a inline function :

```
inline int sum (int a, int b)   → DECLARATION OF INLINE FUNCTION
{ return a+b ; }                → DEFINITION OF THE INLINE FUNCTION
```

Inline functions provide following advantages :

I) Function call overhead doesn't occur.

II) It also saves overhead of a return call from a function.

III) These may be useful for embedded system because inline can yield less code that the function call preamble and return.

Q:2 What is CONSTRUCTOR and DESTRUCTOR ? Explain various type of Constructor ? Is it mandatory to use Constructor in a Class . Justify it .

Ans: Constructors are special methods (functions) that are automatically executed when an object of a class is created. These do not have any return type , not even void and is special because its name is same as the classname.

Destructors are special methods (functions) that releases memory of the Constructors defined .

Constructors are of 3 types :

I) Default Constructor

II) Parameterized Constructor

III) Copy Constructor.

EXAMPLE:

```
class MONEY
{ int rs ;                                  → Data Members
  public : MONEY ();                        // DEFAULT CONSTRUCTOR
           MONEY (int r, int p);            // PARAMETERISED CONSTRUCTOR
           MONEY ( MONEY &M);               // COPY CONSTRUCTOR
           ~ MONEY ();                      // DESTRUCTOR
void show(); void readshow ();              → Member Functions
};
MONEY :: MONEY ()                           } DEFINITION of Default Constructor
{ rs = 0 ; }
MONEY : : MONEY ( int r, int p)             } DEFINITION of Parameterized Constructor
{ rs = r ; }
MONEY : : MONEY (MONEY &M)                  } COPY Constructor
{ rs = M.rs ; }
void MONEY : : readshow ()
{ cin >> rs ;  }  void show ()
{ cout << rs ; }
MONEY :: ~ MONEY ()
{                }
void main ()
{ MONEY m1;
  m1.readshow ();
  MONEY m2 (100);                           → Parameter Passing
  m2.show ();
  MONEY m3 (m2);                            → Copying
  m3.show ();
  getch ();
}
```

Q:3 What the following terms mean :

i) CONSTRUCTOR OVERLOADING :

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different tasks. It is quite similar to the concept of function overloading. Depending upon the number and type of arguments passed, specific constructor is called. Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

ii) ARRAY OF OBJECTS :

The array of type class contains the objects of the class as its individual elements. An array of objects is declared in the same way as an array of any built-in data type. Thus, an array of a class type is also known as an array of objects.

The syntax for declaring an array of objects is

class_name array_name [size] ;

example :

```
class MONEY
{ int rs ;
    public : MONEY () ;                          ] DECLARATIONS OF
             MONEY (int r, int p) ;              ] CONSTRUCTORS
             void readshow () ; void show () ;
};
MONEY :: MONEY ()                                - DEFINITIONS
{ rs = 0; }                                        OF
MONEY :: MONEY (int r, int p)                    - CONSTRUCTORS
{ rs = r ; }
void MONEY :: readshow ()
```

```
{ cin >> rs; }
void MONEY :: show ()
{ cout << rs; }
void main ()
{ MONEY M[2]; int i;
for ( i = 0; i < 2; i++)
{ M[i]. readshow ();
}
MONEY M3 (100);
M3. show ();
for ( i = 0; i < 2; i++)
{ M[i]. show ();
}
getch ();
}
```

DECLARATION OF ARRAY OF OBJECTS

INPUTING IN ARRAY of OBJECTS

DISPLAYING ARRAY OF OBJECTS

Q:4 What is friend function ? Write a program:

Ans: A friend function is a function that is not a member of a class but it has granted all the rights to access private part of a class.
A Member function can also be declared as friend of another class. This is declared using the keyword "FRIEND".

* example:

```
# include <iostream>
# include <conio.h>

using namespace std;
class Person;
class Money
{ int rs; int paisa;
public :
```

→ declaration of class

```
        void read ()
        { cin >> rs >> paisa ; }
        void show ()
        { cout << rs << paisa ; }
DECLARATION
(friend function) ← friend void display ( Person p , Money m ) ;
    };
    class Person
    { char name [10] ;
      public : void input ()
               { cin >> name ; }
               void output ()
               { cout << name ; }
               friend void display (Person p , Money m ) ;
    };
```

While passing parameters →

```
DEFINITION of FRIEND FUNCTION
    void display ( Person p , Money m )
    { cout << p. name << "Contains : " << M.rs << M.paisa ;
    }
    void main ()
    { Money m1 ;
      m1. read () ;
      Person p1 ; p1. input () ;
      display ( p1 , m1 ) ; getch () ; }
```

0:5 Define a class to represent a bank A/c including the following members :

DATA MEMBERS :

a) Name of the depositors
b) Account Number
c) Type of Account
d) Balance amount.

MEMBER FUNCTIONS :

a) to assign initial values
b) to deposit an amount
c) to withdraw an amount after checking the balance.
d) to display the name and balance.

e) WAP to demonstrate the concept of friend function.

Ans:

```
class bank
{ char name [20];
  char accounttype [20];
  int accountno ;
  int amount ;
  int accountbalance ;
  public : void initial ()
           {  name = "NAME";
              accounttype = "SAVINGS";
              accountno = 101 ;
              accountbalance = 0 ;
           }
  void deposit ()
  { cout << "Enter Account Number and
    Account Type for depositing amount : ";
     cin >> accountno >> accounttype ;
    cout << " Enter Amount to deposit";
    cin >> amount ;
     accountbalance = amount ;
  }
  void withdraw ()
  { cout << " Enter accountno to check Balance";
    cin >> accountno ; cout << accountbalance;
    cout << " Enter withdrawal Amount : ";
    cin >> amount;
     accountbalance = accountbalance - amount
  }
friend void display ( bank b)
 { cout << " Enter Account Number to display
   Account Name and accountbalance ;
   cin >> b.name ;
```

```
    cout <<b.accountno <<b.accountbalance ;
    }
};

void main()
{  int choice ;  cin >> choice ;        bank b1 ; b1.initial ;
   cout << " You have entered " << choice ;
   switch (choice)
   { case 1 : b1. deposit ; break ;
     case 2 : b1. withdraw ; break ;
     case 3 : display(b1); break ;
     default : cout << " Enter Correct Choice !! " ;
   }
   getch() ;
}
```

23/11/9

ASSIGNMENT 03

Q:1 What is Inline Function ? Explain with example.

Ans: One of the major objectives of using functions in a program is to save memory space, which becomes appreciable when a function is likely to be called many times.
C++ has solution to eliminate the time of calls to small functions, C++ proposes a new function called inline function.
An inline function is a function that is expanded in line when it is invoked thus saving time.

Syntax :

```
inline function_header ()
{ function_body ; }
```

] SYNTAX

Example :

```
inline void sum ( int a, int b)
{ cout << a+b; }
void main ()
{ int a, b; cin >> a >> b;
  sum ( a, b);
  getch ();
}
```

Q:2 Write a program to describe the concept of Friend function.

Ans:

```
# include <stdio.h>
# include <iostream>      # include <conio.h>
using namespace std;

class Add
{ public : int a; int b;
           void getdata ();
           friend void display (); // declaration of friend function
};
void Add :: getdata ()
{ cout << "Enter two Numbers : "; cin >> a >> b; }
```

```
void display()
{ cout << a+b;    }
int main()
{ Add a;
  a.getdata(); display();
}
```

Q:3 Describe all the Access Specifiers.

Ans: There are 3 access specifiers :

I) **PUBLIC** : members are accessible from outside the class.

II) **PRIVATE** : members cannot be accessed (or viewed) from outside the class.

III) **PROTECTED** : members cannot be accessed from outside the class, however, they can be accessed in inherit classes.

* syntax :

```
class class_name
{ public :                  // public access specifier
        int x;
  private :                 // private access specifiers
        int y;
  protected :               // protected access specifiers
        int z;
};
```

Q:4 Describe all Constructors with an example.

Ans: I) **DEFAULT CONSTRUCTOR** : A constructor that accepts no parameter is called default constructor. This constructor is executed when an object of the class is created.

II) **PARAMETERIZED CONSTRUCTOR** : It is executed when an object of the class is created and is initialized with some

values at the time of creation.

iii) COPY CONSTRUCTOR : this constructor is executed when an object of the class is created and it is initialized with some other objects of the same class at the time of creation.

example :

```
class MONEY
{  int rs;
   public :  MONEY ();          // default constructor
             MONEY (int r) ; // parameterized constructor
             MONEY ( MONEY &M); // copy constructor
             void readandshow ();  void show ();
};
 MONEY :: MONEY()              // definition of Default Constructor
{  rs = 0;  }
MONEY :: MONEY (int r)  // definition of Parameterized
{  rs = r;  }                                  Constructors
MONEY :: MONEY (MONEY &M) // definition of Copy Constructor
{  rs = M.rs ; }                                  MONEY ::
void MONEY :: readandshow ()          void show ()
{  cin >> rs ;   cout << rs ;  }      {  cout << rs ; }
int main ()
{  MONEY M1
   M1. readandshow ();
   MONEY M2 (100);    M2. show ();
   MONEY M3 (M2);    M3. show ();
   getch ();
}
```

Q:5 Write a program to swap private data of two different class

Ans:

```
class b;
class a
```

```
{  int x;
   public : void assign ( int t)
            {   x = t ; }
            void display ()
            { cout << " Value of X is " << x ; }
            friend void swap (a&, b&);
};
class b
{ int y;
  public : void assign (int w)
           {  y = w; }
           void display ()
           { cout << " Value of Y is " << y ; }
           friend void swap ( a&, b&) ;
};
void swap ( a &c , b &d)
{ int temp ;
  temp = c.x ;   c.x = d.y ;   d.y = temp ;
}
int main ()
{  a n ;   b m ;   int aa, bb ;
   cout << " Enter X = " ;   cin >> aa;
   cout << " Enter Y = " ;   cin >> bb ;
   n. assign (aa);   m. assign (bb);
   swap (n, m);
   n. display ();   m. display ();
   getch ();
}
```

} SWAP function

21/11/2019

ASSIGNMENT 04

Q.1 Write a program to find area of Rectangle, circle and square using Function overloading.

Ans:

```
# include <iostream>
# include <conio.h>
using namespace std;

int area (int);
int area (int, int);   float area (float);
int main ()
{   int s, l, b;
    float r;
    cout << " Enter side of a Square :\n"; cin >>s;
    cout << " Enter length and breadth of Rectangle : ";
     cin >> b >> l;
     cout << " Enter Radius of Circle : "; cin >> r;
    cout << "Area of Square is " << area (s);
    cout <<"\n Area of Rectangle is " << area (l, b);
    cout <<"\n Area of Circle is " << area (r);
     getch ();
}
int area (int s)
{ return (s * s); }
int area (int l, int b)
{ return (l * b); }
int area (float r)
{ return (3.14 * r * r);
```

Q.2 What is Constructor and Destructor ? Is it mandatory to use constructor in a class ?

Ans: * CONSTRUCTORS :

These are special class functions which performs initial-

isations of every object. the compiler calls the constructor whenever an object is created. constructor initialize values to object members after storage is allocated to the object.

* **DESTRUCTORS :**
these are special class functions which destroys the object as soon as the scope of object ends. the destructor is called automatically by the compiler when the object goes of out the scope.

* C++ Constructor necessity depends upon class design, requirements. we know that C++ class constructor is called when we create an object of a class. If a class is not required to initialize its data member or does not contain data member, there is no need to write constructor explicitly.

Q:3 Explain Scope Resolution operator with an example.

Ans: Scope resolution Operator is used for following purposes :

I) to access a global variable when there is a local variable with same name,
II) to define a function outside a class,
III) to access a class's static variable,
IV) In case of multiple Inheritance,
V) Refer to a class inside another class. etc.

Example:

```
class A
{ int a;
    public : void readandshow ();
};
void A :: readandshow ()
{ cin >> a ;   cout << a ;
}
```

Q:4 What is static Variable and static function ?

Ans: * STATIC VARIABLE :

they are basically the part of a class instead of a part of the object. they are created in a memory when their class is created. the major difference b/w static and normal variable is that, the static data members are shared among all objects of a class, regardless of the no. of objects because all the objects have an authority to access, modify and change it.

* STATIC FUNCTIONS :

These are the type of members function which could be directly accessed by the name of a class. these are basically the part of a class instead of objects. therefore, only one copy of these member functions is created in a memory with the creation of a class. Sometimes we need to access the member function without creating the objects in a class, in this case static functions are used.

Q:5 Discribe operator overloading .list the operators which cannot be overloaded.

Ans: Operator Overloading is a compile time polymorphism in which the operator is overloaded to provide to special meaning to the user-defined data type.

Operator overloading is used to overload or redefines most of the operators available in C++.

It is used to peform the operators available to operate on the user-defined data type.

the advantage of Operators overloading is to perform different operations on the same operand.

* Operators that can not be overloaded are as follows:

i) Scope Resolution operator (::),
ii) sizeof operator,
iii) Dot or Member operator (.),
iv) Member Pointer Operator (*), and.
v) Ternary Operator (?:)

21/11/2019

ASSIGNMENT 05

Q:1 What is INHERITANCE ? Explain various types of inheritance giving example for each.

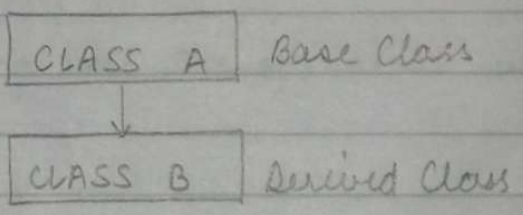
Ans: The capability of a class to derive properties and characteristics from another class is called Inheritance.
Inheritance is one of the most important feature of Object Oriented Programming.

* Sub Class / Derived class : The class that inherits properties from another class.
* Super Class / Base class : The class whose properties are inherited by sub class.

* Types of Inheritance :

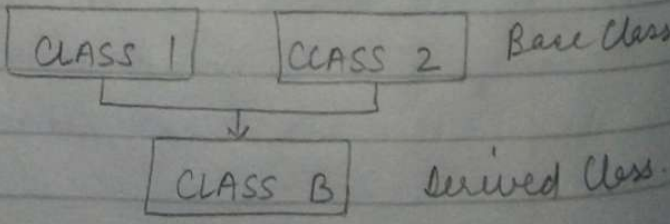
I) SINGLE LEVEL INHERITANCE :
In this, a class is allowed to inherit from only one class, i.e. one sub class is inherited by one base class only.
Syntax :

```
class subclassname : accessmode baseclass
{          };

// A -> B
```



II) MULTIPLE LEVEL INHERITANCE :
In this, we can inherit from more than one classes. i.e. one sub class is inherited from more than one class.
Syntax :

```
class subclassname : accessmode baseclass1 , accessmode
      baseclass 2
{          };

// 1 -> B , 2 -> B
```

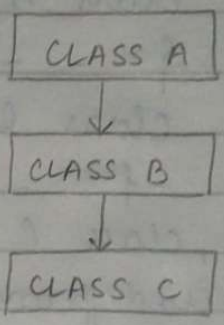


III) MULTI LEVEL INHERITANCE :

In this type of inheritance, a derived class is created from a derived class from a base class.

syntax :

```
class one
{       };
class two : public one
{       };
class three : public two
{       };
```



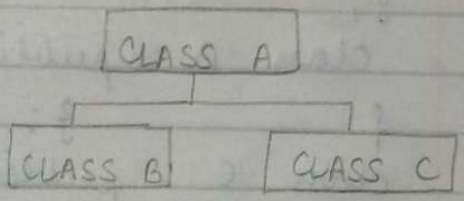
// A -> B, B -> C

IV) HIERARCHICAL LEVEL INHERITANCE :

In this type, more than one sub-classes are inherited from a single base class. i.e. more than one derived class is created from a single base class.

syntax :

```
class one {
};
Class two : public one
{       };
class three : public one
{       };
```



// A -> B, A -> C

Q:2 Explain Multilevel inheritance in detail with suitable C++ Code.

Ans: In this type of inheritance, a derived class is created from another derivet class.

Not only you can derive a class from the base class but you can also derive another class from the derived class.

This form of inheritance is known as Multilevel Inheritance

Example :

* Syntax :

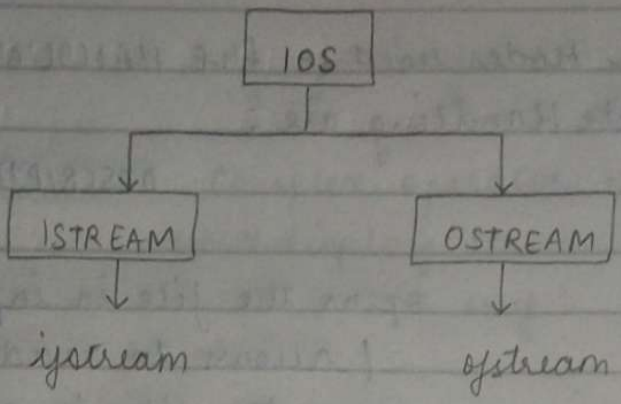
```
class A                          // A
{ - - - };
class B : public A               // A -> B
{ - - - };
class C : public B               // B -> C
{ - - - };
```

Here, class C is derived from class B, and class is derived from class A.

*
```
#include <iostream>        #include <conio.h>
using namespace std;

class A                                  // A
{ public : void display ()
           { cout << "Base Class Content."; }
};
class B : public A                       // A -> B
{          };
class C : public B                       // B -> C
{          };
int main ()
{  C object;
   object.display ();
   getch ();
}
```

Q:3 Explain stream classes with suitable examples.

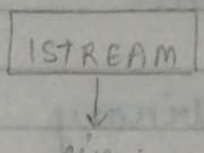
Ans: Stream classes in C++ are used to input and output operations on files and input - output devices. These classes have specific features and to handle I/P and O/P of the program.



1) **ISTREAM Class :**

istream class handles the input stream in C++ programming language. These input stream objects are used to read and interpret the input as a sequence of characters. The CIN handles the input.

In this ifstream is used.



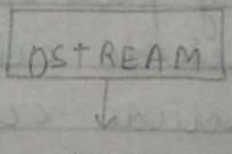
* Example :

```
int main ()
{ int no;
    cin >> no;   cout << no;        // CIN
}
```

II) **OSTREAM Class :**

This handles the output stream in C++ language. These o/p stream objects are used to write data as a sequence of characters on the screen. Cout and puts handles the out stream in C++.

In this ofstream is used.



* Example :

```
int main ()
{ int n;
    cin >> n;
    cout << n;                  // COUT
}
```

Q:4 Define all the modes used in FILE HANDLING.

Ans: Modes in File Handling are:

	MODES	DESCRIPTION
i)	ios :: in	opens the file in input mode (allows to read information)
ii)	ios :: out	opens the file in output mode (allows to write information)
iii)	ios :: app	opens the file in append mode.
iv)	ios :: trunc	deletes the content of a file if the content is present in the file.
v)	ios :: ate	opens the file and moves the pointer to the end of the file.
vi)	ios :: binary	Binary file.

* Default Modes

STREAMS	MODES
a) fstream	- ios :: in - ios :: out
b) ifstream	- ios :: in
c) ofstream	- ios :: out

Q:5 Write a C++ program to add two numbers by overloading binary plus operator (+).

Ans:

```
# include <iostream>
# include <conio.h>
using namespace std;

class complex
{ float x;
  float y;
  public : complex ( float real , float imag )
           {  x= real ;
```

```
        y= imag;
      }
      Complex operator + ( Complex &c);
    void display ()
    { cout << x << y;
    }
};
complex complex :: operator + (complex &c)    //
{ complex temp;
    temp.x = x + c.x;
    temp.y = y + c.y;
    return temp;
}
int main ()
{ Complex c1, c2, c3;
  c1 = complex (2.6, 3.4);
  c2 = complex (7.5, 12.3);
  c3 = c1 + c2;                    // calling of + function
  c1.display ();
  c2.display ();
  c3.display ();
   getch ();
}
```

} DEFINITION OF + OVERLOADED FUNCTION