Unit - IV

Classes and Construction in C++.

Introduction :-

Class is specially introduce new data type in C++. It is a user defined data type. It is similar to structure in C which holding only data while classes hold both data and function.

The only difference between Structure and class is that by default the member of class are private while members of structures are public.

Specifying Class :-

A class is way to bind a data and its associated fun^n together when defining a class we are creating a new abstract data type that can be treated like any other build in data type.

Generally, a class specification has two parts.

1. Class declaration
2. Class function definations.

The class declaration describe the type and scope of its members. The class fun^n defination describes the task of fun^n which is to be implemented.

The general form class declaration is

```
// class declaration
    class classname
      {
        private :
          variable declaration ;
          function declaration ;
        Public :
          variable declaration ;
          function declaration;
      };
```

```
# //Class function defination
    classname :: function name ()
      {
        ——
        ——
        ——
      }
```

The keyword class specifies that it create a abstract data type class name The body of class is enclosed within pair of bresses and terminated by semicolon.

The class body contain the declaration of variable and functions. These functions and variables are collectively called members.

They are usually grouped under two sections namely private and public to denote using the keyword private and public are known as visibility labels.

The keyword followed by colon (:)

The members that have been declared as private can be access only from within the class. On the other hand, public members can be access on outside the class. The data hiding (using private declaration) is the key feature of OOP.

By default the member of class are private.

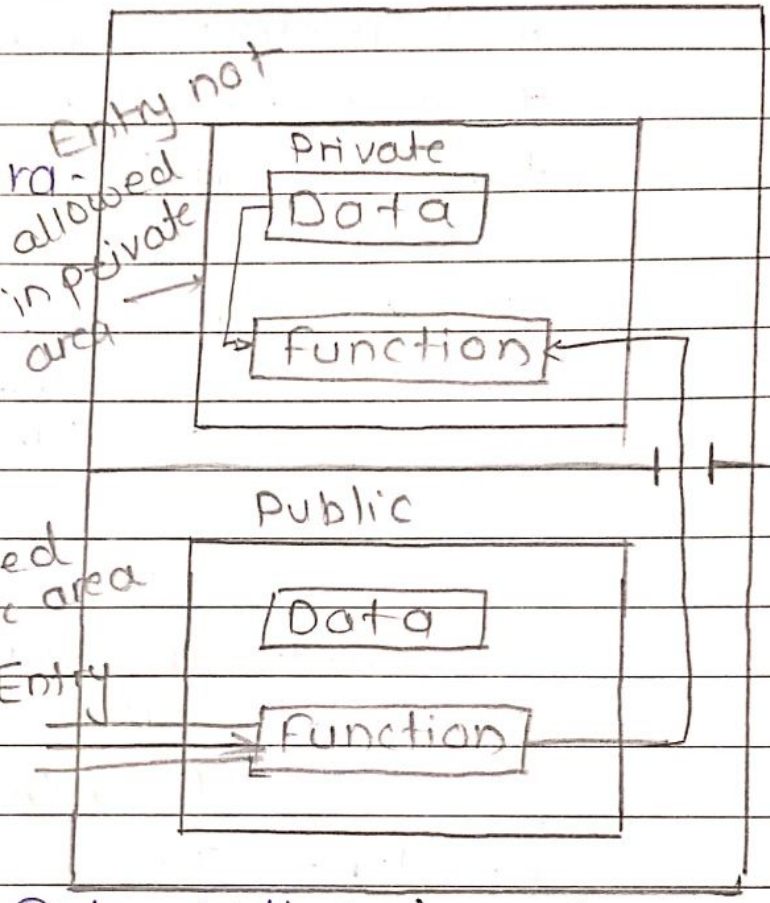
The variables declared inside the class are known as data members and functions are called member function.

Only the member can have access to private data members and private fun^n. The binding of data and fun^n together into a single class type variable is called encapsulation.

A sample class declaration.

e.g.

```
class item
{
    int number;
    float cost;
    Public:
    void getdata();
    void display();
};
```



Data hiding in class

The fun^n getdata() used to assign values to the member variables number and cost and display() fun^n for displaying their values.

Creating Object

Remember that, declaration of item as shown above does not define any object of atom but only specifies what they will contain. Once the class has been declared we can create variable of that type by using the class name.

For example,

```
Item x; // memory for x is
          allocated
```

Create a variable x of type Item. In C++, the class variable are known as objects. Therefore x is called an object of type Item.

We may also declared more than one object in one statement as

```
Item x, y, z; // three objects
                 created.
```

The declaration of an object is similar to that of a variable of any basic type. The necessary memory is allocated to an object at this stage.

We can also create object at the time of class specification also during class defination phase as below,

```
class item
    {
    ____
    ____
    ____
    }
    x, y, z ; // class declaration and
                object creation.
```

Accessing the class members

As pointed out earlier, the private data of class can be access only through the member function of the class. The format for calling member function for the object is as follows :-

object name. function name (argument list)

e.g.

```
x. getdata (5, 100.50); // accessing the
                           getdata fun^n
                           using object x
```

This is valid assignment and assign the value 5 ~~xxx~~ to a number and 100.50 to cost of the object x, after the implementation of getdata function.

Remember, a member function cannot be invoke directly. that is statement like getdata (5, 100.50); has no meaning.

Similarly, the statement like
x. number = 5 is also illegal.
Because here number is data member but it is private data member so cann't be access by the object. It can be access only through member function and not by the object directly.

If variable declared as a public can be access through object.

For example

```
class item
{
  int number ;
  float cost ;
  Public :
  chare name [10] ;
  Void getdata ( int , float) ;
  void display (void) ;
}
```

Defining member function.

A member can be define in two places, Outside the class defination and inside the class defination Irrespective of the place of defination the fun^n should perform the same task therefore the code of the fun^n body would be identical in both the cases.

Only ~~in~~ the difference in fun^n header line.

- Outside the class defination

The Syntax for fun^n defination outside the class is as follows.

```
returntype classname :: funⁿname (arg list)
{
    ———
    ———   funⁿ body.
    ———
}
```

e.g.
```
void item :: getdata (int a, float b)
{
    number = a ;
    cost   = b;
}
```

classname :: tells the compiler that the fun^n name belongs to that class name.

i.e. the scope of fun^n restricted to the class name specified in the header line.

- Function defination inside the class.

While defining the fun^n inside the class it is not necessary to give the scope of fun^n and it is define by replacing the declaration of fun^n in the class.

For eg

```
class item
{
int number;
float cost;
Public:
void getdata (int a, float b)
{
number = a;
float = b;
cost = b;
}
void putdata ( )
{
```

```
    cout << "Number :" << number;
    cout << "\n cost:" << cost;
    }
};
```

20/9/19

* C++ program with class :-

```
// c++ program with class

# include <iostream.h>
 class item
 {
 int number ; // private d.v. by default
 float cost ; // private d.v by default.
 Public :
 void getdata (int a , float b);
                // declaration of fun^n

 Void putdata (void)
    {
     cout << " Number :" << number
          << endl ;
     cout << " cost :" << cost;
    }
};
```

```
void item :: getdata (int a, float b)
{
  number = a;
  cost   = b;
}

void main ()
{
  item x; // create object x.
          x is a variable of type
          item
cout<<"\n object x" <<"\n;"
  x.getdata (100, 99.99);
  x.putdata ();

 item y; // another object
  y.getdata (105, 999.90);
  y.putdata ();
}
```

O/P :-

Object x

number = 100

cost = 99.99

object y

number = 105

cost = 999.90

Date: / /

This program features the class item. This class contents two private variables and two public functions. The member fun^n getdata() which has been defined outside the class which supplies value to both the variable by statement like number = a; cost = b.

This shows that member function can have direct access to private data item.

The another member fun^n putdata() has been define inside the class behaves like Inline fun^n and its task is to display values of the private variable number and cost. The program creates two objects x and y, in two different statements

```
// c++ program with class
# include<iostream.h>
# include <conio.h>
 {
    int RNO;
    Char name [10];   [char * name;]
    float per;
    Public:
    void getdata (int, char, float);
    void putdata (void);
    };

Void student :: putdata ()
```

```
{
  cout << " Object " << endl;
  cout << " R.NO: " << R.NO;
  cout << " Name: " << name;
  cout << " percentage : " << per;
}
void student :: getdata ( int R, char *
                          n, float P)
{
  R.NO = R;
  name = n;
  per  = P;
}
void main ()
   {
    Student S1, S2, S3;
    clrscr();
    S1. getdata ( 101, " Namita", 89.90);
    S1. putdata ();
    S2. getdata ( 105, " kavita", 60.90);
    S2. putdata ();
    S3. getdata ( 110, " Samir", 55.55);
    S3. putdata ();
  getch();
}
```

output :

Object S1
R.NO = 101
Name = Namita
Percentage = 89.90

Object S2
R.NO = 105
Name = Kavita
Percentage = 60.90

object = S3
R.NO = 110
Name = Samir
percentage = 57.55

* Nesting of Member function

A member function can be called by using its name inside another member function of the same class this is called nesting of member function.

The following program illustrates this

// Nesting member function

```
class set
 {
   int m,n;
   public :
   void input (void);
   void display (void);
   int largest (void);
 };
   int set :: largest ()  // defination of
                             member fun^n
                             largest ():
 {
   if (m>=n)
   return (m);
   else
      return (n);
 }
void set :: input (void)
   {
```

```
cout<<" Input values of m & n ; " ;
cin >> m >> n ;
}
void set :: display (void)
{
cout <<" \n largest no is = " << largest()
                <<" \n " ;    // calling
                                member fun
}
main ()
  {
  set A ;  // A is object of class set
  A. input () ;
  A. display () ;

cout<<" \n end of program " ;
}
```

Output :-

```
Input values for m and n
        45  55
largest no. is = 55 :
```

A private member fun^n can only we called by another member function of the same class even an object can not envoke a private function usit dot operator

For ex :-

```
class sample
{
  int m;
  void read (void); // private member
                       fun^n

  public :
   void update (void);

};
S1
S1.update();✓// it works
S1.read(); x // it does not work
```

Array within class :-

The array can be used as member variable in a class, the following class defination is valid.

```
const int size = 10 ; // value of array
                        size

class array
  {
   int a[size]; // a is int type array
                   as data members.

   public :
   void input ();
   void display ();
  };
```

Ex :- // Processing Shopping list

```
# include <iostream.h>
  const int m = 20;
  class item
   char choice
  {
   int item code [m];
   float item price [m];
   public :
    void count ()
     {
      count = 0;
     }
    void get item (void)
      {
      While (choice = 'Y')
      cout << " Enter item code ";
      cin >> itemcode [cout];
      cout << " Enter price ";
```

```
cin >> item price [count];
count ++;
cout << " Enter your choice ";
cin >> choice
}
void display (void);
};
void item :: display (void)
{
 for (int i=0, i<= count; i++)
 cout << item code << "\t";
 cout << item price;
}
main()
 {
  item c1, c2, c3;
     c1. get item ();
     c1. display ();
     c2. get item ();
     c2. display ();
     c3. getitem ();
     c3. display ();
 }
```

O/P :-

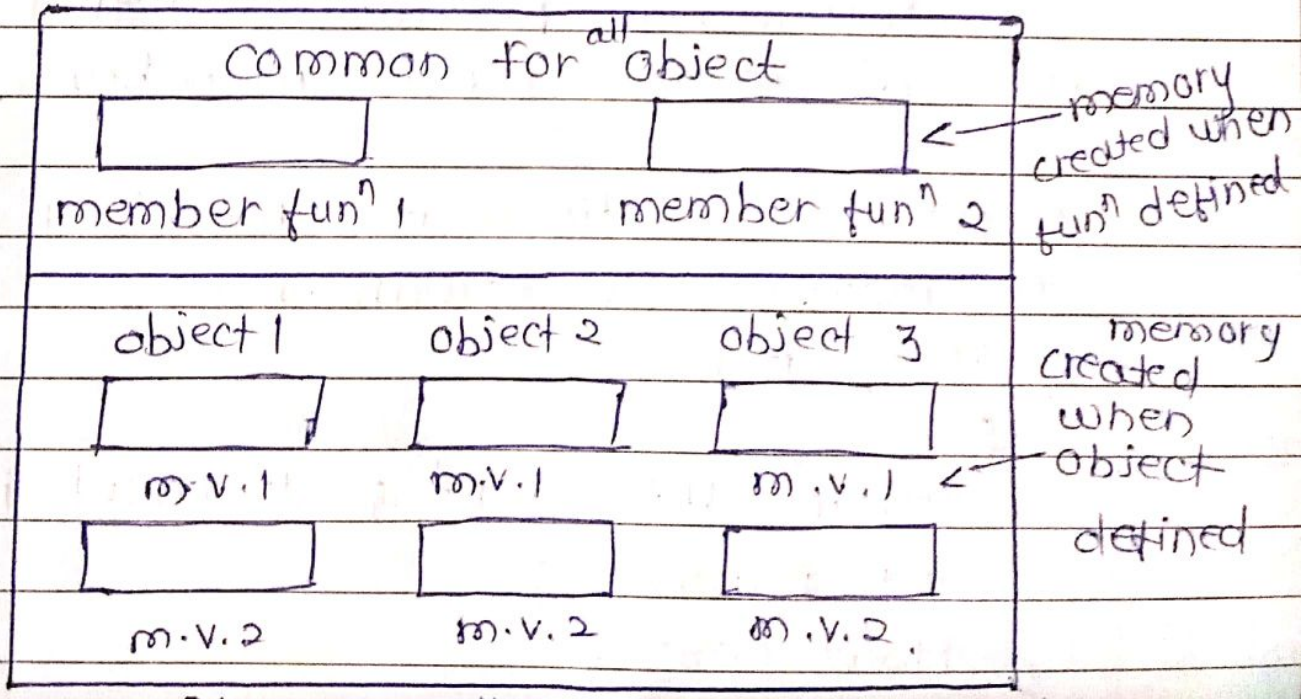
```
item code [1] = 101
item price [1] = 1000
item code [2] = 105
item price [2] = 5000
```

Memory allocation for objects

We started that memory space for object is allocated when they are declared and when not class specified. This statement is partialy true Cpor.

Because the member functions are created and replace in the memory space only once when they are define as part of class specification.

Since all the object belonging that class use the same member fun^{n}. No separate space is allocated for member functions when the objects are created only space for member vaeiables is allocated separately for each object. Separate memory location for the object are essential because the member variable will hold different data for different object.



Memory allocation.

Static data member

A Data members of class can be qualified as static and has the certain special characteristics is it

1) initialize to zero, when the first object of its ~~das~~ class is created
2) only one copy of that member is created for the entire class and is shared as all the object of that class
3) It is visible only with in the class but it's life time is the entire program.

Static variables are namely used to maintain values common to entire class. for ex !-

Counter variable or count variable ma if static that efforts the occurance of all the objects.

following program illustrates the use of static data member.

```
// use of static data member
# include <io stream.h>
 class item
 {
  Static int count ; // count is STATIC
  int number ;
  float cost ;
```

```
Public :
Void getdata (int a, ; ! ! )
 {
    number = a ;
    cost   = b ;
    count ++ ;
 }
 void getcount (void )
  {
  cout << " cout : " << count ;
  cout << " \n number ; " << number ;
  cout << " \n cost : " << cost ;
  }
{ ;
  int item :: count ; // count is defined.
  void main ( )
     {
     Item a, b, c ; // object created
                       count initialize to
     a.getcount(); //  ↘ zero.
     b.get count ();    display count
     C. get count ();
     a. get data (101, 1000 ); // getting data
     b. get data (107, 7000 );      into objects
     c. get data (109, 9000 );
     a. get count ( ) ; // display count
     b. get count ( ) ;
     C. get count ;
```

o/p

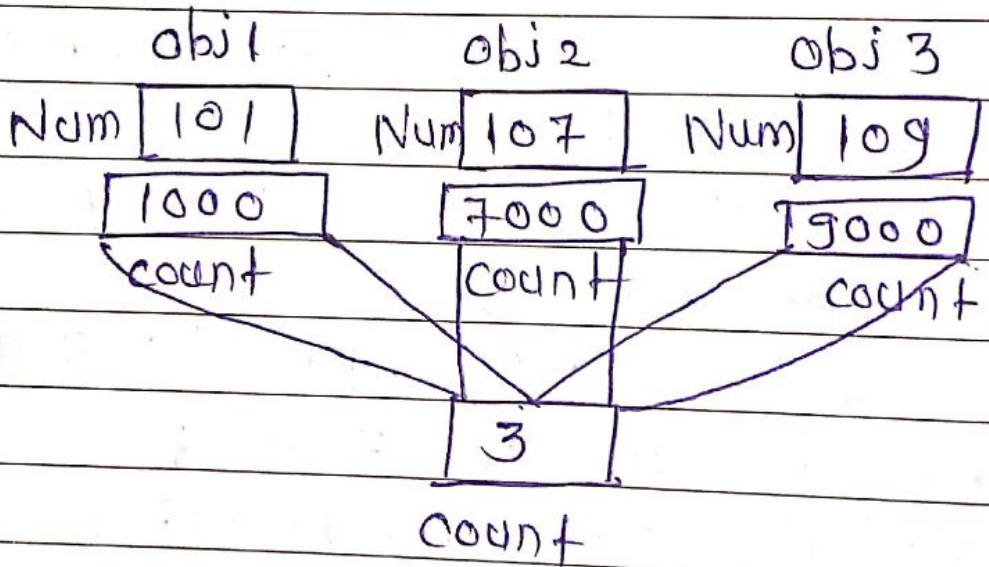
a. number = 101
b. cost = 1000

Count : 0
count : 0
Count : 0

Count : 3
count : 3
count : 3

The static variable count is initialize to zero when the objects are created. The count is incremented whenever the data is read into the object. When the data is read into the object then count will be increase in 3 time.

Since the data is read into the objects 3 times.



Array of object.

We know that an array can be of any data type including type class. Array of variable that are of type class are called array of objects.

Consider, the class defination

```
class Employee
{
 char name[10];
 int age;
 public:
 void getdata (void);
 void putdata (void);
};
```

The identifier employee is a user define datatype & can be used to create objects that relate to different categories of employees

e.g.
```
Employee manager [5];
Employee worker [15];
Employee offices [3];
```

The array manager contains 5 objects.
```
Manager [0];
Manager [1];
Manager [2];
Manager [3];
etc.
```

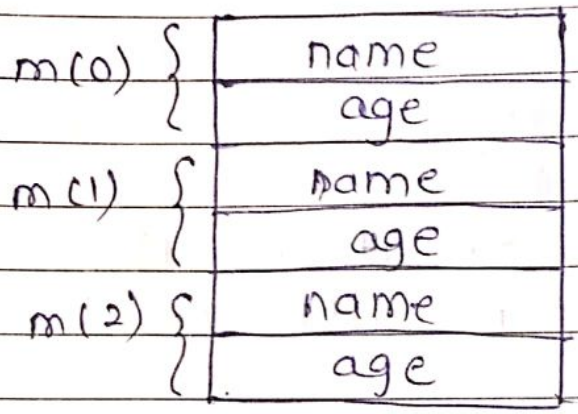
Similarly, Worker array contains 15 object & so on.

We can use usual array accessing method to access individual element & than the dot number operator to access the member function.

for e.g.

Manager [i]. putdata ();

Storage of data item of an object array.



Program to illustrate the use of object arrays

```
# include < iostream.h >
  class employee
   {
    char name [10];
     int age;
    Public :
        void getdata (void);
        void putdata (void);
   };
```

```
void employee :: getdata (void)
 {
 cout<<" enter name";
 cin >> name
 cout <<" enter age";
 cin >> age ;
 }
 void employee :: putdata (void)
  {
  cout<<" name: " << name <<"\n:";
  cout<<" age: " << age ;
  }
Cost int size = 3 ;
void main();
  {
  Employee manager [3] ;
  for(int i =0 ;  i<=2 ; i++ )
  {
  Cout <<" details of manager " <<i+1 ;
   manager [i]. getdata ();
  }

 Cout <<" details of manager " <<i+1;
     manager [i]. putdata ();
 }
```

o/p

Details of manager
Sumit 28
Amit 48
Arun 36
Name : Sumit
Age : 28

Name : Amit
age : 48

Name : Arun
age : 36

21/8/19

Friendly Function :

We know that private members can not be access from outside the class.

i.e A non-member function can-not have in access to private data of an class.

However, there could be a situation where we would like two classes to share a particular function.

In such situation c++ allows the common function to be made friendly with both the classes there by allowing the fun^n to have access to the private

Date: / /

data of these classes. Such a fun^n based not be member of any these classes.

To make an outside fun^n friendly to a class we have to simply declare the fun^n as friend of the class as follows.

```
Class ABC
{
  ===
  ===
  ===

Public :
friend void display ();  //declaration
  ===
  ===
  ===
};
```

The function is defined else where in the program like normal c++ fun^n, without scope operator.

A fun^n can be declared as a friend in any no. of classes and has full access right to the private member of the class. write a program.

The following program illustrates the use of friend fun^n

```
# include < iostream.h >
// use of friend fun^n
  class sample
  {
    int a ;
    int b ;
    Public :
    void setvalue ()
    {
      a = 25 ;
      b = 40 ;
    }
    friend float mean ()
  } ;
    float mean (samples) ;
    {
    return that ((S.a + S.b) / 2.0)
    {
    void main ()
    {
    Sample X ; // object created X
   X . Setvalue () ;
   Cout << " mean = " << mean (X) ;
   {
```

o/p

Mean = 32.5

Note that, the friend fun^n access the class variables a & b by using the dot operator and the object passed through it.

The fun^n call mean (x) passes the object x by value to the friend fun^n. Member fun^n of one class can be friend fun^n of another class. In such cases they are defined with scope operator.

```
Class X
{ ____
  ____
  int fun1(); // member function
  ____
  ____
  ____
}
class Y
{ ____
  ____
  ____
Friend int x :: fun1(); //
  ____
  ____
  ____
}
```

Static Member function

A member fun^n that is declared with keyword static is called static member function which has following properties.

① A static fun^n can have access to any other static members declared in the same class

② A static member function can be called using the class name instead of its object as follows

Class name :: static fun^n name ();

The following program illustrates the concept of static member fun^n.

```
# include <iostream.h>
  class test
    {
     int code;
     static int count ; // static mem. variable
     public :
     void getcode (void)
      {
      code = ++count ;
      }
      Void showcode (void)
      {
       cout<<" object number :" << code;
```

```
    {
    static void showcount(void) // static
      {                                  member fun'
      cout<<"\n count : "<<count;
      }
};
    int test :: count;
    main()
    {
      test t1, t2;
        t1. setcode();
        t2. setcode();
      test :: showcount(); // accessing static
                                fun^n with class
    }                             name
      test t3;
        t3. setcode();
      test :: Showcount();
        t1. showcode();
        t2. Showcode();
        t3. showcode();
```

Output :-

```
Count 2
Count 3
object NO. 1
object NO. 2
object NO. 3
```

Constructors

Constructor is a special member function whose task is to initialize the object of its class. It is special because its name is same as that of class name.

The constructor is envoke whenever an object of the class is created. It is called constructor because it construct the values of data member of class.

e.g.

```
// class with constructor
class integer
  {
    int m, n;
    Public :
    integer (void) ; // constructor declared
    ____
    ____
    ____
  }

integer :: integer (void)
  {
    m = 0;
    n = 0;
  }
```

In the declaration of object in main program as follows.

```
main()
{
  integer I1 ; // object I1 created & construction invoked
  ---
  ---
}
```

This declaration not only creat the object I1 but also initializes its data member n and m to zero & this statement invoke the constructor fun^n automatically.

The constructor fun^n has some special characteristics

1) They are always declared in public section
2) They invoke automatically when the object created.
3) They do not have any return type not even void.
   Therefore, they cannot return values
4) They cannot be inheritate
5) They can have default arguments
6) Constructor cannot be virtual.

Parameterized Constructor.

The constructor integer( ) defined above initializes the data members of all the object to zero.

But in practice it is necessary to initialize the various data element of different objects with different values when they are created. c++ allows us acheive this object by passing argument to the constructor when the objects are created. Such constructors are called parameterized constructors.

The following example illustrates this.

```
class integer
{
  int m, n ; // data variables
  Public :
  integer (void); // constructor declared
  integer ( int , int); no argument
                        // parameterized
                           constructor)
};
integer :: integer (void)
  {
    m = 0 ;
    n = 0 ;

  }
```

```
integer :: integer(int x, int y)
{
    m = x;
    n = y;
}

main()
{
    integer I1; // constructor I is
                    invoked.
    integer I2(5,10); // const "
    integer I3(15,20); // "
```

O/P :-

```
I1.m = 0
I1.n = 0
I2.m = 5
I2.n = 10
I3.m = 15
I3.n = 20
```

There are two methods for calling constructor

```
integer I1 = integer(5,10);
        explicit call

integer I1(5,10);
        implicit call
```

Remember that, when a constructor is parameterized we must provide appropriate arguments for the constructor.

Following program illustrates passing of arg. to the constructor function.

```
# include < iostream.h>
   class integer
     {
       int m, n;
       public :
        integer (int , int ); // Parameterized
        void display ()          constructor
        {                          declared
        cout<<" m = " << m <<"\n ";
        cout <<" n = " << n <<"\n";
        }
      integer :: integer( int x , int y)
        {
           m = x;
           n = y;
        }
      void main ()
        {
         integer int (0, 100);
                          // implicit call

        integer int2 = integer ( 25, 75);
                          // explicit call
```

```
cout<<"\n object 1 : " <<"\n";
int 1. display ();
cout <<"\n object 2 : " <<"\n";
int 2 . display ();
```

O/P

object 1 :

m = 0 n = 100

object 2 :

m = 25 n = 75

Multiple constructors in a class

It is possible to declare multiple constructors in a class. and they are differ from one another in the number of arguments.

According to the arguments value supplied during the object creation the respective constructors are invoke by matching the no. of arguments.

```
# include <iostream.h>
  class complex
   {
     float x, y;
     Public :
     complex () { } // const.1 no argument
     complex (float a) // const.2
     {
        x = y = a ;
     }
    complex (float real, float imag)
     {
        x = real ;
        y = imag ;
     }
    friend complex sum (Complex , complex)
     friend void show (complex) ;
     };

    complex sum (complex &1 ; complex (2)
     {
       complex c3 ;
       c3.x = c1.x + c2.x
       c3.y = c1.y + c2.y
       return c3 ;
     }
    void show (complex c) // friend
     {
     cout << c.x << " j " << c.y << "\n"

     }
```