Date : 05/07/19
Page No.:

2. Classes And Object.

- **Classes -**

Class is the collection of object with similar properties.

Class is the collection of data members and member functions.

The entire set of data and code can be made user defined datatype with the help of class.

The class is way to bind data and functions to allow the data to be hidden. is necessary to external use.

The class specification has two parts -

① class declaration.

It describes the type and scope of its variable.

② class function defination -

It describes how do class are implemented.

Syntax -

```
class classname
{
    private :
      variable declaration ;
    public :
      function declaration ;
};
```

Date : / /
Page No.:

The keyword "class" specifies that what follows is an abstract data type.

The body of a class is enclosed within a curely brauses and terminated by semicolon.

The class body contains delaration of variable and function.

The function and variable are collectively called as class members.

They are usually grouped under two sections, namely private & public, to denote which of the members are private and which are public.

The keyword "private" and "public" are also known as "visiablity levels"

Note-

Private and Public keywords are followed by colon (:).

The class members that have been declaraed as private can be accessed only within the same class.

On the other hand public members can be accessed from outside the class.

The use of keyword private is optional, by defaut the member of the class are private.

If bothe the lables are missing then by defaut all the members are private.

Date : / /
Page No.:

Such class is completely hidden from outside the world and does not serve any purpose.

The variable declared inside the class are known as "data members" and the function declared inside the class are known as "member functions"

The binding of data and functions together into a single class type variable is refered as "Incapulation".

eg-

```
class student
{
  private :
   int roll_no;
   char name[20];
   float percentage;

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

student is class name. The class student contains three data members and two member functions.

The data members are private and member functions are public.

The function get() can be used to assign the values to the member variables

roll_no, name, percentage and display() data function display their value.

- creating object -

once the class has been declared we can create variables of that type by using class name.

In c++ the class variables are known as "objects".

We may also declare more than one object in one statement.

Syntax -

classname class_type V_1, class_type V_2 ..... class_type V_n ;

eg-

student s1, s2, s3, ..., sn ;

- Accessing data members.

The private data of class can be accessed only by "using member function"

Syntax -

Objectname . function_name(Argument list);

Date : / /
Page No.:

eg-

```
s1.getdata();
s1.displaydata();
```

Imp

program = (inside the class defination)

```
#include <stdiostream.h>
#include <conio.h>
class student
{
  private :
  int roll_no;
  char name[20];
  float percentage;

  public :
  void getdata()
  {
  cout<<" enter roll_no, name &
  percentage";
  cin>>roll_no>>name>>percentage;
  }

  void display_data()
  {
  cout<<"roll_no=" << roll_no;
  cout<<" name=" << name;
  cout<<" pecentage = " <<percentage;
  }

};
```

```
void main()
{
   Student  s1, s2, s3;
    s1.getdata();
    s1.displaydata();
    s2.getdata();
    s2.displaydata();
    s3.getdata();
    s3.displaydata();

    getch();
}
```

- Defining the member functions.

The member functions can be defined in two ways:

① Inside the class defination.
② outside the class defination.

① Inside the class defination.

Inside the class method defining the ~~method~~mber function is to replayce the function declaration by the actual function defination inside the class.

eg-

```
class  items
{
  private:
```

```
int item_no;
char item_name[20];
float item_price;

public:
void getdata()
{
  cout<<"enter item_no, item_name,
  item_price";
  cin>>item_no>>item_name>>item_
  price;
}
 void displaydata()
 {
  cout<<"item_no = "<<item_no;
  cout<<" item_name = "<<item_name;
  cout<<" item_price = "<<item_price;
 }

};
void main()
{
  items i1, i2;
  i1.getdata();
  i1.displaydata();
  i2.getdata();
  i2.displaydata();
  getch();
}
```

Date : / /
Page No.:

② Outside the class defination.

Member function that are declared inside the class have to be defined separatly outside the class.

The Defination is very much like a normal function.

An important difference in member function and normal function is that member function is incorporates a membership identity lable in the header.

This lable tells the compiler which class the function belongs to.

Syntax -

```
return_type classname :: functionname (list of arguments)
   {
      body of function
   }
```

eg.

program

```
#include <iostream.h>
#include <conio.h>
class student
{
  private :
    int roll_no;
    char name[20];
    float marks;
```

```
public :
  void getdata();
  void displaydata();
};

void student :: getdata()
{
   cout<<" enter roll_no, name, marks";
   cin>>roll_no>>name>>marks;
}

void student :: displaydata()
{
   cout<<"roll no"<<roll_no;
   cout<<" name"<<name;
   cout<<" marks"<<marks;
}

void main()
{
  student s1, s2;
     s1.getdata();
     s1.displaydata();
     s2.getdata();
     s2.displaydata();

     getch();
}
```

Date : / /
Page No.:

- Characteristics of Member function

① Several different classes can use the same function name.

② Member function can access the private data of the class, a non-member function cannot do.

③ The member function can call another member function directly without using dot (.) operator.

- Nesting of member function.

We can call member function from another class member function.
This can be nesting of member functions, we can also declare member functions as a private member of a class.

- program -

```
# include <iostream.h>
# include <conio.h>
class student
{
  private:
   int Roll_no;
   char name[20];
   float marks;
   public:
```

```
    void getdata();
    void displaydata();
    void showdata();
};
void student :: getdata()
{
    cout<<"enter RollNo, name, marks";
    cin>>RollNo>>name>>marks;
}
void student :: showdata()
{
    cout<<"RollNo="<<RollNo;
    cout<<"name="<<name;
    cout<<"marks="<<marks;
}
void student :: displaydata()
{
    cout<<"Displayed data="<<showdata();
}
void main()
{
    student s1,s2;
    clrscr();
    s1.getdata();
    s2.getdata();
    s1.displaydata();
    s2.displaydata();
    getch();
}
```

```
# include <iostream.h>
# include <conio.h>
Class  number
{
  private :
    int  num1, num2 ;

  public :
    void  read ();
    void int  max ();
       void  show_max ();
  };

Void  number :: read ()
{
  cout<<" enter the num1 value ";
  cin >> num1 ;
  cout<<" enter the num2 value";
  cin >> num2;
}

int  number :: max ()
{
    if ( num1 > num2)
     {
       return  num1 ;
     }
    else
     {
       return  num2 ;
     }
```

```
}
void number :: show_max ()
{
    cout << " maximum number = " << max();
}
getch();
}
  void main()
{
    number n1, n2;
    clrscr();
     n1.read();
     n1.max();
     n1.show_max();
     n2.read();
     n2.show_max();
     getch();
}
```

Date : / /
Page No.:

- private member function.

private member function are the functions which are declared inside the private section.

Private member functions are the function which are assigned inside the other functions i-e the functions which are present in public sections.

eg - program:

```
#include <iostream.h>
#include <conio.h>
Class number
{
  private :
int num1, num2;
  int max();

  public :
  void read();
  void show_max();
};
void number :: read()
{
  cout << "number 1 value is = " ;
  cin >> num1;
  cout << " number 2 value is = " ;
  cin >> num2;
}
```

```
int number :: max ()
{
    if (num 1 >= num 2)
    {
        return num 1;
    }
    else
    {
        return num 2;
    }
}
void number :: showmax ()
{
    cout<<" maximum number = "<< max();
}
void main ()
{
    number n1, n2;
    clrscr();
    n1.read ();
    n1.showmax();
    n2.read ();
    n2.showmax();
    getch();
}
```

(4m)

V.V Imp.

Memory allocation for object

Once we define class it will not allocate memory space for the data member of the class.

The memory allocation for the data member of the class is performed separately each time when an object of class is created.

When an object of a class is created the amount is equal to the amount of memory required by members of that class.

This is because each object member of a class has its own copy of the data members. All the object of a class share the same copy of the data members and member functions of a class.

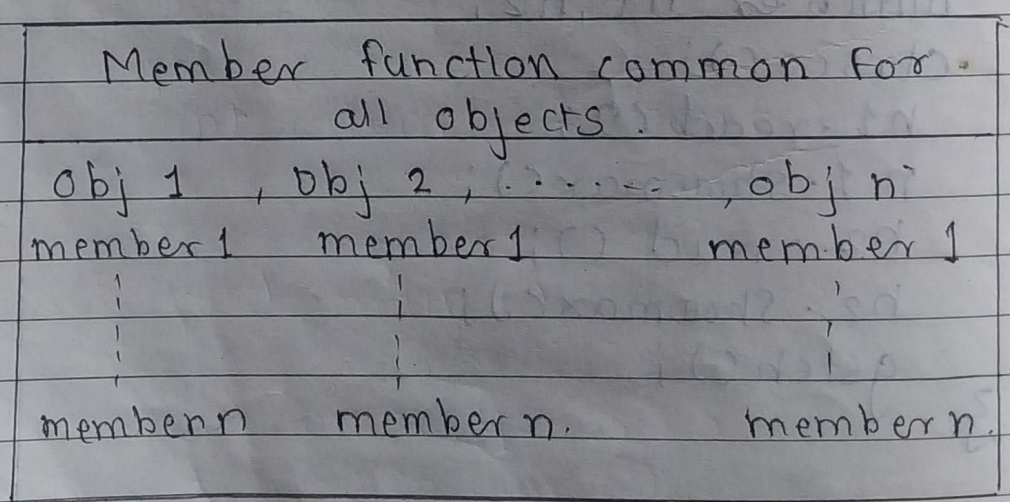

fig - objects in memory.

Date : / /
Page No.:

UI-I

V.V Imp

- Static data members

A data member of a class can be made static.

static variable is initalized to zero (0) When the objects are created.

When the member variable declaration is preceded with keyword "static".

It tells the compiler that one copy of that variable will exists and all object of class will share that variable.

- Special characteristics of Data members.

① It is inisalized to zero when the object is created. No other mitalization is perimated.

② only one copy of that member is created for entire class and is shared by all the objects of that class (same class), no matter how many objects are created.

③ Static variables are normally used to maintain the values common to all entire class.

④ The type and the scope of each static member variable must be defined outside the class defination.

Syntax -

```
Datatype classname :: Static variable name;
// Static member defination
```

eg - program -

```
# include <iostream.h>
# include <conio.h>
class item
{
    private:
     static int count;
     int no;

    public:
    void getdata (int a)
    {
       no = a;
       count ++;
    }
    void getcount ()
    {
       cout << " count = " << count;
    }
};
int item :: count; // Static member defination.

void main ()
{
    item x1, y1, z1;
    x1.getcount ();
    y1.getcount ();
    z1.getcount ();
    x1.getdata (100);
    y1.getdata (200);
```

```
    z1.getdata(300);

    cout<<" After reading data";
     x1.getcount();
     y1.getcount();
     z1.getcount();

     getch();
}
```

output-

0
0
0

After reading data.

3
3
3

Static member function

The static member function can have access only other static members declared in the same class.

Static member function can be called using the class name instead of its object.

syntax-

```
class name :: static_function_name();
```

Date : / /
Page No.:

Static member does not have "this" pointer.

program -

```
#include <iostream.h>
#include <conio.h>
class test
{
    private :
     int code ;
     static int count ;

    public :
     void setcode ()
    {
        code = ++count ;
     }
    void showcode ()
    {
      cout <<" object no = " << code ;
    }
    static void showcount ()
    {
      cout <<" count= " << count ;
    }
};
int test :: count ; // static member defination

void main ()
{
    test t1 , t2 ;
```

```
t1.setcode();
t2.setcode();
test::showcount(); // accessing
static member function.

    test t3;
    t3.setcode();
    test::showcount();
     t1.showcode();
     t2.showcode();
     t3.showcode();

    getch();
}
```

Output -

```
count = 2
count = 3
object no = 1
object no = 2
object no = 3
```

Date : / /
Page No.:

- Array of object -

Array of variable that are of the type class is called as Array of object.

The dot (.) operator is used to access the member functions, the index is used to represent the elements of array.

The array of object is stored inside the memory in the same way as a multi dimensional array.

Syntax = classname objectname[size];

program -

```
# include <iostream.h>
# include <conio.h>
class employee
{
  private :
  int age;
  char name[20];

  public :
  void getdata();
  void displaydata ();
};
  void employee :: getdata()
  {
    cout <<" enter age, name of employee";
    cin >> age >> name;
  }
```

Date : / /
Page No.:

```
void employee :: displaydata()
{
  cout<<"age ="<<age;
  cout<<"name="<<name;
}
void main()
{
  employee e[5]; int i;
  clrscr(); ⟶ for(i=0;i<5;i++)
  {e[i].getdata();
   e[i].displaydata();
  }
  getch();
}
```

V.V IMP

Q1 Write a program for class student having datamembers name and rollno for 4 objects. using Array of object (outside the class defination.

→

```
#include <iostream.h>
#include <conio.h>
class student
{
  private :
    int Roll_no;
    char name[20];

  public :
    void getdata();
    void displaydata();
};
```

Date : / /
Page No.:

```
void student :: getdata ()
{
    cout << " enter the name and roll_no
    of student ";
    cin >> name >> Roll_no ;
}

void student :: displaydata ()
{
    cout << " name = " << name ;
    cout << " Roll_no = " << Roll_no ;
}
void main ()
{
    int i ;
    student s[4] ;
    clrscr ();
    for(i=0 ; i<4 ; i++)
    {
        s[i].getdata ();
        s[i].display_data ();
    }
    getch();
}
```

Date : / /
Page No.:

3 times. V Imp.

• Object as a function arguments.

In c++ we can pass object as a function argument and also return an object from the function.

An object can be passed to a function as an arguments. in the same way as the variable of any other data type, In this process the copy of object is passed to the function. It can be done by in two ways-

① Pass by value
② Pass by reference.

① Pass the value-

copie of entire object is passed to the function, any changes made to the ~~function~~ object inside the function do not affect the object used to call the function.

eg- program.

```
# include <iostream.h>
# include <conio.h>
class sample
{
    private :
      int x;

    public :
      void getdata()
    {
        cout << " enter the value of x ";
        cin >> x;
```

```
};
 value
 void displaydata()
 {
    cout<<" value of x = "<<x;
 }

 void swap (Sample S1, sample S2)
{
        int temp;

        temp = S1.x;
        S1.x = S2.x;
        S2.x = temp;

 cout<<"Swaped value of S1 of x = "<<S1.x;
 cout<<"swaped value of S2 of x = "<<S2.x;
};

 void main()
{
  Sample S3, S4, S5;
   clrscr();
     S3.getdata();
     S4.getdata();
     S3.displaydata();
     S4.displaydata();
     S5.swap(S3, S4);
     getch();
};
```

Date : / /
Page No.:

② Pass by reference.

When an address of the object is passed, the called function works directly on the actual object used in the call, this means that any changes made to the object inside the function call will reflect in the actual object.

program -

```
# include <iostream.h>
# include <conio.h>
class sample
{
    private:
     int x;

    public:
    void getdata()
    {
        cout<<" enter the value of x";
        cin>>x;
    }

    void display data()
    {
        cout<<" value of x = "<<x;
    }

    void swap( sample *s1, sample *s2)
    {
        int temp;
        temp = s1->x;
```

```
    s1->x = s2->x ;
    s2->x = temp ;

cout << "swaped value of s1 of x = " << s1->x;
cout << "swaped value of s2 of x = " << s2->x;
  }

 void main()
{
    sample s3, s4, s5;
      s3.getdata();
      s4.getdata();
      s3.displaydata();
      s4.displaydata();
      s5.swap(&s3, &s4);
      getch();
  }
```

compulsory. V.V. IMP

- Friend function.

The private members cannot be accessed from outside the class, i.e non-member function. cannot have access to the private data of that class.

Sometimes there could be a suitation where we would like to share a private member to non-member function.

In such cases c++ allows a common function to be made friendly with both the classes there by allowing the function to have access private data of this classes.

Date : / /
Page No.:

To make outside the function friendly to a class simply declare this function as a friend function.

syntax-

```
class  class_name
{
    private :
        ----
        ----

    public :
        ------
        -------

        friend  void  function_name ();
    // friend function declaration.
};
```

The function defination does not use either the keyword "friend" or "scope resolution operator (::)"

The function which are declared as "friend" with the keyword "friend" is known as "friend function."

The function can be declared as a friend in any number of classes.

The friend function is member function has full rights to access private member of the class.

Date : / /
Page No.:

characteristics of friend function.

① It is not in scope of the class in which it has been declared as a friend.

② Since, it is not in the scope of the class it cannot be called by using the object of that class.

③ It can be invoked (to call) like a normal function without the help of any object.

④ Unlike the member function it cannot access the member names directly and has to use the object name and dot (.) operator.

eg -

```
temp = a1.x + a1.y;
```

⑤ It can be declared either in public or private part of the class without affecting the meaning.

eg - Program -

```
# include <iostream.h>
# include <conio.h>
class Sample.
{
```

Date : / /
Page No.:

```
private :
 int a, b;

public :
 void getdata()
{
    a = 10;
    b = 20;
}
Friend sample mean(sample s);
{; // friend function declaration.

int/sample mean (sample s)
 {
    return (s.a + s.b) / 2;
 }

 void main ()
{
    sample s1; int x;
    clrscr();
    s1.getdata();
     x = mean(s1);
    cout<<" mean value = "<< x;
    getch();
 }.
```

Date : / /
Page No.:

program=

```
#include <iostream.h>
#include <conio.h>
class myclass_2 ; // forward declaration
class myclass_1
{
  private :
   int a;

  public :
   void getdata ()
  {
   cout<<" enter one value = ";
   cin>>a;
   cou
  }
   Friend int max (myclass_1 m1,
  myclass_2 m2);

 };

class myclass 2
{
  private :
    int b;
  public :
  void getdata ()
  {
    cout <<" enter another value = ";
    cin >> b;
  }
```

Date : / /
Page No.:

```
Friend  int max (myclass_1 m1, myclass_2
m2);
};

int max(my_class_1 m1, myclass_2 m2)
{
    if (m1.a > m2.b)
    {
        return (m1.a);
    }
    else
    {
        return (m2.b);
    }
}
void main ()
{
    myclass_1  m1;
    myclass_2  m2; int x;
    clrscr();
    m1.getdata();
    m2.getdata();
    x=max(m1, m2);
    cout<<" maximum number = "<<x;
    getch();
}
```

Date : / /
Page No.:

FIX
V.V.IMP

Constructor and distructor

Introduction -

While implementing the classes members function are used to provide the initial values to the private member variables.

All the function call statements to member functions are used with appropriate objects that has already been created.

This function cannot be used to initalize the member variables at the time of creation of their objects.

A class is user defined datatype, it should behave very similar inbuilt data type.

This means that we should be able to initalize the class type variable (objects). When it is declared the same way as the initalization of and ordinary variable

ex -

```
int a = 10;
```

But this can be done with ~~constructor~~ special member function called as "Constructor".

Defn :-

Constructor is special member function whose task is to initalize the object of its class.

It is special because it's name is same as the class name.

Date : / /
Page No.:

The constructor is envoked (to call) whenever an object of it's class is created.

Syntax - eg -

```
class time
{
  private :
    int h,m,s;

  public :
    time ();  // constructor function declaration
};

time :: time ()// constructor fun def^n
{
   h=0;
   m=0;
   s=0;
}
void main ()
{
   time t1;
   getch ();
}
```

Time t1 is not only create the object but also initalize the private data members i.e - h=0, m=0, s=0.

Date : / /
Page No.:

4th. Characteristics of constructor -

① They should be declared in public section.
② They are envoked automatically when the objects are created.
③ They do not have return types, not even void and therefore they cannot return any value.
④ They cannot be inherited through a derived class can called the base class constructor. even they are declared as friend function.
⑤ like other c++ function they can have default arguments.
⑥ It cannot refer that address.

Imp. Types of constructor -

① Default constructor :-
A constructor that accepts "no parameters", is called as default constructor.
The default constructor for class employee,

```
emp :: emp() // constructor function
                defination.
emp e; // constructor function
          calling.
```

Date : / /
Page No.:

envoke default constructor when object is created.

program :-

```
# include <iostream.h>
# include <conio.h>
# include <string.h>
class sample
{
  private:
   int cost;
   char name[20];
  public:
   sample ();  // constructor declaration
  void display ();
};
sample :: sample ()
{
   strcpy (name, "scale");
    cost = 30;
}
void sample :: display ()
{
   cout << "the item name=" << name;
   cout << "the item cost " << cost;
}
void main ()
{
  sample s1;
  clrscr();
  s1.display ();
  getch();
}
```

Date : / /
Page No.:

② Parameterized constructor :-

It is possible "to pass arguments" to constructor function, typically this arguments help to initalized an object when it is created.

Sometimes it is essential to initalize the various data element of different objects with different values when they are created. this is achieved by passing arguments to constructor function when the objects are created.

The constructor which accepts any number of formal parameters & that formal parameters that are used to initalize the object is called parameterized constructor.

A constructor which has one or more parameters then this function is called as Parameterized constructor

Syntax-

```
class class_name
{
  ---
  ---

  public:
  class_name(parameter1, parameter2
  ---par n);
    {

    }
};
```

Date : / /
Page No.:

program -

- (Implicit calling)

```
#include <iostream.h>
#include <conio.h>
class addition
{
  private:
    int x, y, z;

  public:
    addition (int a, int b)
    {
      x=a;
      y=b;
    }
    void display()
    {
      z=x+y;
      cout <<" value of z = " << z;
    }
};
void main()
{
  addition P(10, 20); // implicit calling
  clrscr();
  P.display();
  getch();
}
```

Date : / /
Page No.:

V.V Imp. 17,18.
W-16, Y-13, S-16

3) overloaded or multiple Constructor.

C++ allows defining multiple constructor with different number, datatype in a signal class using constructor overloading.

The number of constructors can be defined for the same class with various argument list as referd as overloaded "constructor or multiple constructor".

eg -

program -

explicit =

```
# include <iostream.h>
# include <conio.h>
class addition
{
  private:
   int a, b, c;
 public:
 addition (); // default constructor
                declaration
 addition (int x, int y) // parameterized
                           constructor
                           declaration
};
addition :: addition ()
{
    a = 10;
    b = 20;
}
```

```
addition :: addition (int p, int q)
{
    a=p;
    b=q;
}
void addition :: display()
{
    c=a+b;
    cout<<" value of c ="<<c;
}
void main()
{
    addition a1;
    addition a2 = add (5,15) // explicite calling
    a1.display(); // o/p = 30
    a2.display(); // o/p = 20.
    getch();
}
```

4) Copy Constructor :-

This is anathor special type of constructor is copy constructor that is capable of creating a new object as replace of existing once.

The copy constructor is constructor which creates an object by initalizing it with an object of same class which has been created priviously, in another words the copy constructor takes an argument which is an object of same class. Initalize

one object from another of same type.
A copy of an object to pass it as an argument to a function.

syntax -

```
function_name (userdefined_datatype _ &object)
void main()
{
  sampe s1(s2); // copy constructor calling
    OR
  Sample (s1 = s2); // copy constructor calling.
```

e.g.

program -

```
#include<iostream.h>
#include<conio.h>
class sample
{
  private:
  int a, b;
  public:
  Sample()
  {
    a=10;
    b=20;
  }
Sample (sample &s) // copy constructor
{
  a=s.a;
  b=s.b;
}
vold display()
```

Date : / /
Page No.:

```
{
  cout << " value of a = " << a;
  cout << " value of b = " << b;
  }
};
void main()
{
   sample s1; // default constructor calling.
    s1.display();
    sample s1(s2); // copy constructor calling
          or
    Sample (s1 = s2); // copy constructor calling.
    s2.display();
    getch();
  }
```

Q. Write a program class name is date & data members are day, month, year with copy constructor accept for 1 and display the result for two object.

→

```
#include <iostream.h>
#include <conio.h>
class date
{ int d,m,y;
  public:
  date();
  {
    d = 1;
    m = 3;
    y = 2019;
  }
```

```
date (date &d) // copy constructor
{
    d=d.d;
    m=d.m;
    y=d.y
}

void display()
{
  cout<<" Day=" <<d;
  cout<<" month =" <<m;
  cout<<" year =" <<y;
   {
};

void main()
{
    date d1; // default constructor
    d1.display();
    date &d1 (d2);
          or
     date d1 = d2;
     d2.display();
    getch();
}
```

Output -

Day 1, month 3, year 2019

Day 1, month 3, year 2019

Date : / /
Page No.:

5. Dynamic constructor -

The allocation of memory to object at the time of their construction or their object.

The constructor can also be used to allocate the memory while creating the Object.

This will enable the system to allocate the right amount of memory for each objects when the objects are not of the same size.

program=

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class sample
{
  private:
  int len;
  char *name;

  public:
  sample(char *s)
  {
    len = strlen(s);
  name = new char[len+1];
   strcpy(name,s);
  }
  void display()
  {
```

```
    cout<<"name="<<name;
    }
};
void main()
{
  Sample S("SJT");
      or
  Sample S = Sample("SJT");
   S.display();
   getch();
}
```

* Constructor with default arguments -

It is possible to define constructors with default arguments.

eg - Constructor.

complex(float real, float imag = 0)

The default of argument imag is '0' then the statement will be

= complex C1(10.0, 20.0);

Assign first value 10.0 to real variable and second vale 20.0 is to the imag variable.

The actual parameters when specified overwrites the default value.

The missing argument must be the "training value"

eg -

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

Date : / /
Page No.:

```
class complex
{
  private :
  float real, imag;

  public :
  complex (float x , float y=0)
  {
      real = x;
      imag = y;
  }

  void display()
  {
     cout << " value of real = " << real;
     cout << " value of imaginary = " << imag;
  }
};

void main()
{
   complex c1(10,20); // Implicit.
   ~~clrscr();~~   OR
    c1.display();
   complex c2(10);
   c2.display();
   complex c3 = complex(10,20);
    c3.display();
    getch();
  }
```

Imp. declare a class SI, data members are principle amount, rate of interest and no. of years, the constructor will have default value of rate of interest as 11.5 %.

Accept this data for two objects, calculate and display S.I of each object.

→

```
# include <iostream.h>
# include <conio.h>
class SI
{
  private :
   int amount;
   float r_of_I;
   float year, SI;

  public :
  SI(float a, float y, float r=11.5)
{
        princ
        amount = a;
        year = y;
        r_of_I = r;
}

void display()
{
    SI = (amount * year * r_of_I)/100;
    cout << "Simple interest = " << SI;
}
```

```
};

void main()
{
    SI   s(10000, 2); // Implicit
    s.display();
    SI  S1 = SI (10000, 3); //Explicit.
     S1.display();
     getch();
    }
```

Q Roll_no, name, course, class student. course (char.) as default.

→
```
# include <iostream.h>
# include <conio.h>
# include <string.h>
 class student
{
  private:
   int roll_no, length;
   char  *course, *name;

   public:
   student (int r, char *n, char *c = "IT")
   {
      Roll_no = r;
      length = strlen(n);
      length = strlen(c);

      name = new char [length+1];
      course = new char [length+1];
```

```
    strcpy (name, n);
    strcpy (course, c);
}

  void display()
{
   cout << endl << "rollno =" << roll_no;
   cout << endl << "course =" << course;
   cout << endl << "name =" << name;
  }
};
void main()
{
   student s(07, "shreya");
   clrscr();
   s.display();
   getch();
}
```

output =
rollno = 7
course = IT
name = shreya.

Date : / /
Page No.:

Fix

Explain Distructor with eg.

* Distructor :-

A distructor is a spectal member function of a class.

A distructor is used to destroy the objects that have been cretted by constructors.

The distructor is a special Member fuction whose names is same as class name. But it is preceded by a symbol. tilde (~)

eg -

```
class Sample
{
  private :
    int m,n;

  public :
  sampe( ); // Default constructor
               function declaration.

  ~ sample ( ); // distructor function
                   declaration.
};
```

A distructor neither takes any arguments nor return any value.

It will be invoked implicitely by the compilar upon exit from the program to cleanup the storage. i.e no longer accessable.

• Explain distructor with example ?

Date : / /
Page No.:

Distructor is called at the end of the program by the compiler.

eg -

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

int count =0 // Global variable
                declaration.
Class alpha
{
 public:
  alpha ()
  {
    count ++;
    cout <<" Number of objects are created:
    " << count;
  }
 ~ alpha () // distructor defn.
{
    cout <<" Number of objects are
    distroyed:" << count;
     count --;
  }
};
 Void main ()
{
   cout <<" Enter in main ";
   alpha a1, a2, a3, a4;
     {
        cout <<" enter in Block no 1";
```

```
    {
        alpha a5;
    }
{
    cout << "enter in block NO 2";
    alpha a6;
    }
    cout << "Return in main";
  } getch();
}
```

output :-

```
enter in main
Number of objects are created : 1
Number of objects are created : 2
Number of objects are created : 3
Number of objects are created : 4

enter in Block no 1
Number of objects are created : 5
Number of objects are distroyed : 5

enter in Block no 2
Number of objects are created : 5
Number of objects are distroyed : 5
Return in Main.
Number of objects are distroyed : 4
Number of objects are distroyed : 3
Number of objects are distroyed : 2
Number of objects are distroyed : 1.
```

Program –
Class add declare three variable a, b, c.

```
#include <iostream.h>
#include <conio.h>
class add
{
   private:
    int a, b, c;

   public:
    add (int p, int q) // Parameterized
   {                      constructor funct.
      p=a=p;
      q=b=q;
   }
   void display ()
   {  c=a+b;
    cout<<" Addition = "<<c;
   }
   ~add ()
{
   cout<<" All the objects are distroy
    ed :";
   }
};
  void main()
{
   add a1 (10,20); // Implicit calling
   clrscr();
   a1. display ();
   getch();
}
```

Date : / /
Page No.:

output

Addition = 30

All the objects are distroyed.

--- *** ---