The Shirpur Education Society's
R. C. Patel College of Engineering & Polytechnic, Shirpur
Department of Computer Science & Engineering

Course Title - Object Oriented Programming Using C++

Course Code - 313304

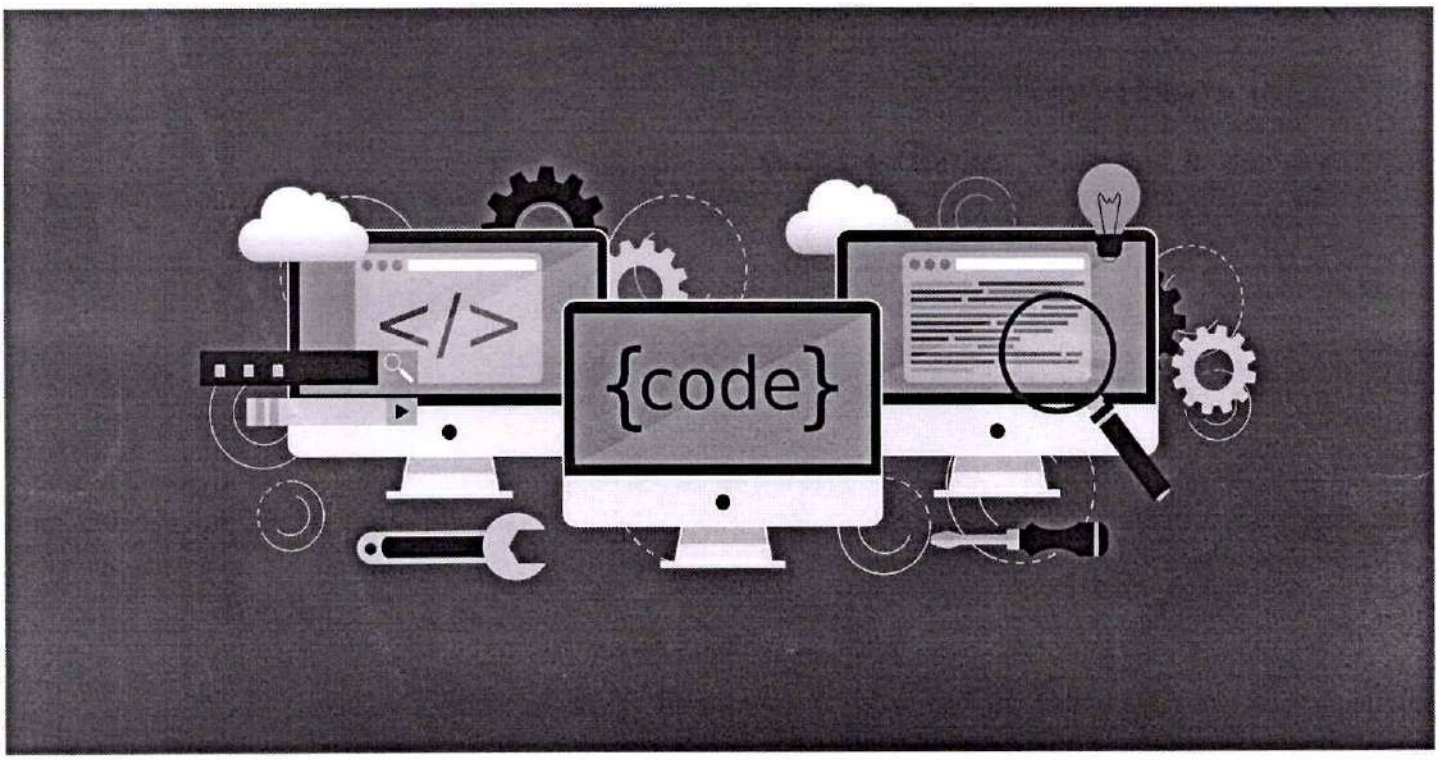
Programme Name - Computer Science & Engineering

Semester - Third

Unit - II Functions and Constructors

Total Marks:16

Prepared By: Mr. Sudhakar B. Baviskar



Unit-2 M.-16

Functions And Constructors

2.1 Inline Function :- An Inline Function is a Function in which the compiler Replace the Function Call With the Actual Function Code to Make Execution Faster

An inline Function is declared using the inline keyword

Syntax :-

```
Inline Return_type Function_name
                              (Parameter)
{
    // Function body.
}
```

Where,

inline → keyword Used to Make Function Inline

Return_type → Type of value returned by function like int, Float, void... etc.

Function name → Name of the Function

Parameter → Values passed to the function

function body → Code Written inside the function

Example :-

```
#include <iostream.h>
Using namespace std;
inline int square (int x) // Inline
{                          Function declaration
return x*x; // return Square of no.
```

```
}
    int main()
{
    int num=5;
cout<<"square of"<<num<<"="<<square(num);
                        //calling inline fun.
    return 0;
}
```

o/p: square of 5=25

Advantages:-

1. Faster Execution
2. Easy to use
3. Better for small operations
4. Saves Function calling time

Disadvantages:-

1. Increases program size
2. More Memory Usage
3. Not suitable for large function

Uses of Inline function:-

1. Increase Execution speed
2. Reduce function call overhead
3. Execute small functions quickly
4. Improve program performance

Difference between Inline Function & NON-Inline Function

Keypoints	Inline Function	NON-Inline Function
Keyword	Use inline keyword	No inline keyword
Execution	function code is substituted directly	function call is resolved at runtime
Performance	faster	slower
Binary size	increase due to code duplication	smaller as code is not duplicated.

Static Data Member:-

Static Data Members are class Member declared using the static keyword.

A static data Member is a variable that is shared by all objects of a class

Characteristics of Static Data Members:-

1. only one copy of a static data Member exists for the entire class, shared by all object
2. It is initialized once & exists independently of class object
3. Static data Member is Automatically Initialized with zero.
4. Each Static variable must be defined outside the class defination
5. Static data Member Must be defined outside the class definition & Stored seperately from the instanee (Non-static) data Member

Syntax:

```
class class_Name
{
    Static data_type variable_name;
};
```

Example: 1) if 100 students are in 1 college College Name is same for all students So college Name can be static

2) A Static data Member can be Used as counter that records the occurrences of All the objects.

Program:-

```
#include<iostream.h>
Using namespace std;
class item
{
    static int count;
    int number;
public:
    void getdata(int a)
    {
        number = a;
        count++;
    }
    void getcount(void)
    {
        cout<<" count";
        cout<< count <<"\n";
    }
};
int item::count;
int main()
{
    item a, b, c; // count is initialized to zero
    a.getcount(); // display count
    b.getcount();
    c.getcount();
    cout<<" count
    a.getdata(100);
    b.getdata(200);
    c.getdata(300);
cout<<" After Reading data"<<"\n";
```

```
    a.getcount();
    b.getcount();
    c.getcount();
    return 0;
}
```

O/P:-

```
Count : 0
count : 0
Count : 0
```

After Reading data

```
Count : 3
Count : 3
Count : 3
```

. The Static Variable Count is Initialized to zero when the object are created the Count is incremented whenever the data is Stored / Read into An Object Since the data is Read into object 3 times, the Variable Count is incre. mented 3 times because there is only 1 copy of Count Shared by all the three objects, All the 3 objects O/P Statement cause the value 3 to be displayed

figure (1) shows how a Static Variable is Used by the objects

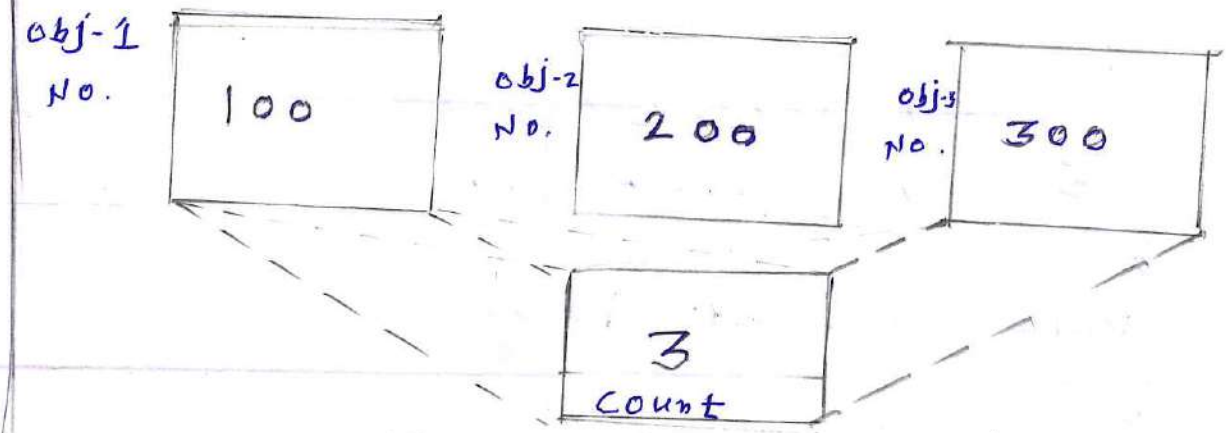


fig. (1) Sharing of a Static data Member

* Static Member Functions:-

A Member function that is declared Static has the following properties

- A Static function can have Access to only Static Members (function or variable) declared in the same class
- A Static Member function can be called using the class name (instead of it's objects) As follows.

```
class_name :: function_name;
```

In below example The Static function showcount() display the number of objects Created till that Moment A count of No. of object created is Maintained by the Static variable count

The function Showcode() displays the code number of each object

program:-

```
#include<iostream.h>
using namespace std;
class test
{  int code;
   static int count; // static Member variable
public:
   void setcode(void)
   {  code = ++count;
   }
   void showcode(void)
   {
```

```
    cout << "object number :: " << code << "\n";
}

    static void showcount(void) // static member function
{
    cout << "count : " << count << "\n";
}
};
    int test :: count;
    int main()
{
    test t1, t2;
    t1.setcode();
    t2.setcode();
    test :: showcount();
    test t3;
    t3.setcode();
    test :: showcount(); // Accessing static function

    t1.showcode();
    t2.showcode();
    t3.showcode();

    return 0;
}
```

O/P

```
Count : 2
count : 3
object number : 1
object number : 2
object number : 3
```

* Friend Function Using two different classes:

- Friend function can be used to access the private & protected Members of two or more different clasess this is achieved by declaring the function as a friend within each of the clasess whose private / protected Members it need to Access

Syntax:-

```
class ABC
{
  ------
  ------
 public: ---
   friend void xyz(void); //Declaration
};
```

Charactristics:-

1. it is Not in the Scope of the class to which it has been declared as friend
2. It can be invoked like a Normal function without the help of Any object
3. It can be declared either in the public or the private part of a class

Program:-

```
#include<iostream.h>
using namespace std;
class sample
{
    int a;
    int b;
public:
    void setvalue() { a=25, b=40;}
friend float Mean(sample s);
};
```

```
float Mean(Sample S)
{
    return float (S.a+S.b)/2.0;
}
    int main()
    {
        Sample X;
        X.SetValue();
    Cout<<"Mean Value ="<<Mean(X)<<"\n";
        return 0;
    }
```

* Friend Function Using NON-Member Function

A Friend function is a Normal function (NON-Member) that can access the private & protected Members of a class when it is declared Using the keyword "Friend"

Syntax:

```
class className
{ friend return_type function_name
                    (class name object);
};
```

program:

```
#include<iostream.h>
Using namespace std;
class Demo
{
Private:    int num;
public:
        Demo()
        {
            num=50;
        }
```

```
    Friend void show (Demo d);
};
    void show (Demo d)
{
    cout << Number =" << d.num;
}
    int main()
{
    Demo obj;
    show (obj);
    return 0;
}
```

o/p :- Number = 50

2.2 Array of objects :-

An array of objects is a collection of objects of the same class stored in continious memory location

It allows us to handle Multiple objects using a single array name

Syntax :-

```
ClassName object_name(size);
```

Ex :- `Student S[5];`

program :-

this creates 5 objects of class student.

```
#include <iostream.h>
Using namespace std;
class student
{
    private:
        int rollno;
        char name[20];
    public:
```

```
Void getData()
{
 cout << "Enter Roll No";
 cin >> rollno;
 cout << "Enter Name";
 cin >> name;
}
 void showData()
 {
  cout << "\n Roll No = " << rollno;
  cout << "\n Name = " << name;
 }
};
  int main()
 {
  Student S[3];
  int i;
  for(i=0; i<3; i++)
 {
  cout << "\nEnter Data for Student" << i+1 <<
                                        endl;
   S[i].getData();
  }
   for(i=0; i<3; i++)
   {
    cout << "\n Student" << i+1;
    S[i].showData();
   }
   return 0;
 }
```

O/p Enter Data for Student 1
Enter Roll No. 1
Enter Name : Amit
Enter Data for Student 2
Enter Roll No. 2
Enter Name : Rahul
Enter Data for Student 3

Enter Roll No. 3
Enter Name: Gitesh

Object Array Declaration:-

Student S[3];

Creates 3 objects

S[0], S[1], S[2]

Accessing objects

S[i].getData();

S[i] → particular object

. → Member Access Operator

getData() → Member function call

* **Object As Function Arguments:-**

When an object of a class is passed to a function as an Argument, it is called object as Function Argument

The Function can use the object data Member & Member functions

Syntax:-

Function_name (object_name);

Program:-

```
#include<iostream.h>
Using namespace std
class Demo
{   int num;
public:
        Void getData()
{       cout<<"Enter Number";
        cin>>num;
}
```

```
void show(Demo d)
{
    cout<<"Number="<<d.num;
}
};
int main()
{
    Demo d1, d2;
    d1.getData();
    d2.show(d1);
    return 0;
}
```

O/P Enter Number : 10

Number = 10

Function Call

d2.show(d1);

d1 object is passed As Arguments.

2.3 Constructor

A constructor is a special Member function of a class that is Automatically called when an Object is Created.

Characteristics of constructor:

1. Constructor Name is Same as class name
2. They do not have Return type
3. They should be declared in public Section
4. They are invoked Automatically when the object are created.
5. Can not be overloaded
6. Can not be static

7. can not be inherited.

8. Used to initialize objects.

Syntax:-

```
class className
{
  public:
  className()
  {
     //constructor body
  }
};
```

Program:-

```
#include<iostream.h>
using namespace std;
class student
{
   int roll;
Public:
     student()   //Constructor
     {
       roll=101;
     }
     Void show()
{
     Cout<<"Roll No="<<roll;
};
};
int main()
{
    Student s1;
    s1.show()
    return 0;
}
```

O/P:-

Roll No.= 101

Constructor Declaration

Student()

Student → Constructor Name

Same As class Name &

No Return type

Object creation:-

Student S1;

When object S1 is created, constructor is called Automatically.

*Types of Constructors

1 Default Constructor

2 Parameterized constructor

3 Copy constructor

1. Default Constructor:-

A default constructor is a constructor that has No arguments (parameter)

It is Automatically called When an Object is created

Syntax:-

```
class ClassName
{
  public:
  ClassName()
  {
    constructor body
  }
};
```

Program:-

```
#include<iostream.h>
Using namespace Std;
class Student
{
    int roll;
public:
        Student()   // Default constructor
    {     roll=101;
     }
        void show()
      {
     Cout<<"Roll NO="<<roll;
  }};
     int main()
   {
        Student S1; // Constructor Called Automatically
        S1.show();
        return 0;
     }
```

O/P:-

Roll NO.=101

2. Parameterized Constructor:-

A Parameterized constructor is a constructor that takes one or more Arguments (parameter) It is used to initialize object data Members with User-defined values at the time of object creation.

It is used to Initialized object with different values.

Syntax:-

```
class className
{
    data-type Variable;
public:
    className ( parameter List)
    {
        // initialization
    }
};
```

Program:-

```
#include<iostream.h>
Using namespace std;
class Student
{   int roll;
    int Marks;
public:
    Student ( int r, int m)
    {
        roll = r;
        Marks = m;
    }
    void show()
    {
        cout<<"Roll No = "<<roll<<endl;
        cout<<"Marks = "<<Marks<<endl;
    }
};
int main()
{
    Student S1(101, 85);
    S1.show();
    return 0;
}
```

O/P = Roll No = 101
Marks = 85

Difference between Default Constructor & Parameterized Constructor in C++

Default constructor	Parameterized constructor
1. Has No Arguments	Has Arguments
2. Initializes object with fixed value	Initialized object with user defined values
3. Called without parameter	Called with parameter
4. Less flexible	More flexible
5. Used for Default initialization	Used for Dynamic initialization
6. Same values for All objects	Different values for different objects
7. Syntax: Student()	Syntax: Student(int r)

3. Copy Constructor: A Copy Constructor is a constructor used to create a new object by copying the data of Another object of the same class

- Constructor that takes a Reference to Another object
- Initializes a New object as a copy of an existing object.

Syntax:

```
className (className & objectName)
{
}
```

Program:

```
#include<iostream.h>
Using namespace std;
class Demo
{
    int num;
public:
    Demo(int n)  // parametrized constructor
    {
        num=n;
    }
    Demo(Demo &d) // copy constructor
    {
        num = d.num;
    }
    void show()
    {
    cout<<"Number ="<<num<<endl;
    }
};
int main()
{
    Demo d2(d1); // copying object
    d1.show();
    d2.show();
    return 0;
}
```

O/P

```
Number=10
Number = 10
```

2.4 Constructor overloading

* Constructor overloading allows a class to have Multiple constructor with the same Name (which is class Name) but with different parameter List & it is quite Similar to function overloading

Key aspects of Constructor overloading :-

1. Multiple constructors: A class can define More than one Constructor
2. Same Name: All overloaded Constructors Must have the same name as the class itself
3. different parameter Lists: Constructor differ by 1. Number of Arguments 2. Type of Arguments 3. Order of Arguments.

Syntax:

```
class className
{
    public:

    className(); //Default constructor
    className(int x); //parameterized const^n
};
```

Program:

```
#include<iostream.h>
 Using namespace std;
class Demo          //class declaration
{
public:
    Demo()          //Default constructor
    {
```

```
    cout<<" Default constructor called "<<endl;
}

    Demo(int x)   // constructor overloaded here
{   cout<<" parameterized constructor called"<<
                                        endl;
    cout<<"value of x = "<<x<<endl;
};  }
    int main()
{   Demo d1;    // calls default constructor
    Demo d2(10); // calls overloaded constructor
    return 0;
}
```

o/p

```
Default constructor called
Parameterized constructor called
value of x = 10
```

* **Constructor With Default Arguments:**

A Constructor with default arguments is a Constructor where default values are assigned to parameter

if No value is passed during object creation the default value is used Automatically

Syntax:

```
ClassName(int x=10)
{
=
}
```

Program:-

```
#include<iostream.h>
  Using namespace std;
  class Box
{
   Public: int length, width, height;
  Box(int l=5, int w=10, int h=2)
{
   length = l;
   width = w;
   height = h;
}
   void display()
{
  Cout<<"Length:"<< length<<", width:"<< width
  <<", Height:"<<height<<endl;
  }
};   int main()
   {
  Box box1;
  Box box2(15);
  Box box3(15, 20);
  Box box4(15, 20, 25);
     box1.display();
     box2.display();
     box3.display();
     box4.display();
     return 0;
}
```

Destructor :- A Destructor is a Special Member function Used to destory objects Created by the constructor

It is Automatically called when an object goes out of scope or the programs ends.

Characteristics of Destructor:

1. Destructor Name is Same as class name
2. Destructor start with ~ symbol
3. No Return type
4. No Arguments are passed
5. Automitically called
6. Used to free Memory & Resources
7. Used to destory object
8. Destructor can-not be overloaded.
9. Destructor can not be static
10. Destructor can not be inherited
11. Destructor can be Virtual

Syntax:

```
~ className()
{
}
```

Program:-

```
#include<iostream.h>
Using namespace std;
class Demo
{
public:
    Demo()
    {
```

```
    cout<<"Constructor called"<<endl;
}
    ~Demo()
{
    cout<<"Destructor called"<<endl;
}
};
    int main()
{
    Demo d1;
    return 0;
}
```

O/P:-

Constructor called
Destructor called.

* Difference between Constructor & Destructor

Sr No.	Constructor	Destructor
1.	Used to Initialize object	Used to destory object
2.	Constructor Name is Same As class Name	Destructor Name is same as class Name with ~ symbol.
3.	Automatically called when object is created	Automatically called when object is destory
4.	Can be overloaded	Can-not be overloaded
5.	can take Arguments	Does Not take Arguments.

Sr No	Constructor	Destructor
6.	Used for Memory Allocation	Used for memory deallocation
7.	Helps in objects setup	Helps in cleanup work.
8.	Syntax: className()	Syntax: ~className()

Shaikh 25/05/26

25/5/26.

Unit-II

Functions & Constructors

OOP - 313304

Question Bank

1. Define Constructor, List it's types (W-2025) 2 Marks
2. State Any two Rules to define friend function (W-2024) 2 Marks
3. Define destructor, State Any two features of destructor (S-2025) 2 Marks
4. Describe Use of Static data Member (S-2019) 2 Marks
5. Explain Inline Member function (W-2023) 2 Marks
6. State Any two Characteristics of Constructor (W-2024) 2 Marks
7. State Any two characteristics of destructor (W-18) 2 Marks
8. Explain friend function with Suitable Example (S-2025) M-04
9. Define Static Member function with Syntax & State the properties of Static Member function (Any two) (S-2024) M-4
10. Explain constructor Overloading give Suitable program code in C++ (W-2024) M-04
11. Explain Parametrized Constructor with Suitable Code example (W-2024) M-04

12. State the difference between Constructor & Destructor (Any 6 points) (S-2023) M-06

13. Define copy Constructor & explain it's use with example (W-2025) M-04

14. W. A. program in c++ that includes, Constructor & destructor (W-2025) M-04

15. Develop a c++ program for constructor with default Argument & use of destructor (W-2022) 6M.