UNIT-3

CO:
Demonstrate Constructors, destructors &
Operator-Overloading.

Introduction to Constructors: Characteristics, Constructor with Default Arguments, Parameterized Constructors, Overloading Constructors, Copy Constructor, Dynamic Constructors and Destructors, Anonymous Objects. Introduction to operator Overloading, Rules for Overloading Operators, Overloading Unary & Binary Operators, this keyword, Constraint on Increment and Decrement Operators, Overloading with Friend Functions, Type Conversions.

1.Characteristics of *Constructors*

- Constructor have the same name as that of the class they belongs to.
- Constructors must be declared in public section.
- They automatically execute whenever an object is created.
- Constructors have neither return value nor void.
- The main objective of constructor is to initialize objects and allocation of memory for objects.
- Constructors are executed implicitly, they can be invoked explicitly.
- Constructors have default values and can be overloaded.
- A constructor without any arguments is called as default constructor.
- A constructor takes arguments is called as parameterized constructor.

```
1)class num
{
        int a, b;
public:
        num()
        {
        a =5; b=2;
        }
        void show()
        {
        cout<<a<<b;
        }
};
void main()
{
num n;
n.show();
}
```

```
2)class num
{
        int a, b;
public:
        num()
        {
        Cout<<“Enter two values:”;
        cin>>a>>b;
        }
        void show()
        {
        cout<<“The given two values
are:”<<a<<b;
        }
};
void main()
{
num n1,n2;
n1.show();
n2.show();
}
```

2. Constructor with Default Arguments

- Similar to member functions, It is possible to define constructors with default arguments.
- Consider power(int n, int p= 2);
  - The default value of the argument p is two.
  - power p1 (5) assigns the value 5 to *n* and 2 to *p*.
  - power p2(2,3) assigns the value *2* to n and 3 to *p*.

```
class power
{
        int b,p;
public:
        power(int n=2, int m=3)
        {
        b=n;
        p=m;
        cout<<pow(b, p);
        }
};
void main()
{
        power x;
        power y(5);
        power z(3, 4);
 }
```

Output:

8 125 81

3. Parameterized Constructors

- It may be necessary to initialize the various data elements of different objects with different values when they are created. This is achieved by passing arguments to the constructor function when the objects are created.
- The constructors that can take arguments are called parameterized constructors.
- They can be called explicitly or implicitly.

```
class num
{
int a, b ,c;
public:
num(int x, int y)
{
a=x; b=y;
}
void show()
{
cout<<a<<b;
}
};
void main()
{
num n(10,20);//implicit call
n.show();
num x= num(1, 2);//explicit call
 x.show();
}
```

Output:

```
10 20
1          2
```

4. Overloading Constructors

Similar to normal member functions, constructors also overloaded. C++ permits to use more than one constructors in a single class.

If a class contains more than one constructor. This is known as constructor overloading.

Add( ) ; // No arguments
Add (int, int) ; // Two arguments

Example:

```
class num
{
int a, b;
public:
num()       // Default constructor
{
a =10; b=20;
}
num(int x, int y) // Parameterized  constructor
{
a=x; b=y;
}
void add()
{
cout<<a+b;
}
};
void main()
{
 num n;
n.add();
num x(1,2);
x.add();
}
```

5. Copy Constructor

Copy constructor is used to declare and initialize an object from another object.

A copy constructor takes a reference to an object of the same class as itself as an argument.

```
class Test
{
int i;
public:

Test()      // Default constructor
{
i=0;
}
Test (int a)// Parameterized constructor
{
i = a;
}
Test (code &x)       //Copy Constructor
{
i = x.i;
}
void show()
{
cout<<i<<endl;
}
};
 void main()
{
Test a(100);
a.show();
Test b(a);//Copy Constructor invoked
 b.show();
}
```

Destructor

- Destructor is a special member function like a constructor. Destructor destroy the objects that are created by constructors.
- It is automatically executed when the object goes out of scope.
- Destructor releases memory space occupied by the objects.

Characteristics of Destructors

- Their name is the same as the class name but is preceded by a tilde(~).
- They do not have return type, not even void and they cannot return values.
- Only one destructor can be defined in the class.
- Destructor neither has default values nor can be overloaded.
- We cannot refer to their addresses.
- They make "implicit calls" to the operators ***new*** and ***delete*** when memory allocation/ memory de-allocation is required.
- Programmer cannot access addresses of constructors and destructors

- Constructors and destructors cannot be inherited, but a derived class can call the constructor and destructors of the base class.
- Destructor can be virtual, but constructors cannot.

Example:

```
class Test
{
    public:
    Test()
    {
    cout<<“\n Constructor Called”;
    }
    ~Test()
    {
    cout<<“\n Destructor Called”;
    }
};
 void main()
{
Test t;
}
```

6. Dynamic Constructors and Destructors,

- The constructor can also be used to allocate memory while creating objects. Destructor destroy the memory.
- This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size,thus resulting in the saving of memory.
- Allocation of memory to objects at the time of their construction is known as dynamic construction of objects.
- The memory is allocated with the new operator.
- The memory is destroyed with the delete operator.

```
#include<iostream>
using namespace std;
class dyncon{
            int *ar;
            int n;
public:
            // default constructor
            dyncon();
            //destructor
            ~dyncon();
};
dyncon::dyncon()
{
   cout<<"Dynamic Array creation &
                initialization:"<<endl;
    cout<<"enter size of the array:";
    cin>>n;
    ar=new int[n];
    cout<<"enter elements:";
for(int i=0;i<n;i++)
cin>>ar[i];
}
dyncon::~dyncon()
{
   cout<<"\nBefore Destroy the
              element:"<<ar[2];
   cout<<"\nAfter destroy the Array:";
   delete ar;
   cout<<ar[0]<<endl;
}
int main()
{
            dyncon obj1;
            return 0;
}
```

```
#include<iostream>
#include<string.h>
using namespace std;
class dyncon{
        char *name;
        int len;
        public:
        // default constructor
        dyncon()
        {
        len=0;
        name= new char[len+1];
        name[0]='a';
        }
        dyncon (char *s)
        {
        len=strlen(s);
        name= new char[len+1];
        strcpy(name,s);
        }
        ~dyncon()
        {
        delete name;
    cout<<"object destroyed"<<name[1]<<endl;
        }
        void display()
        {
        cout<<name<< endl;
        }
};

int main()
{
dyncon obj1;
obj1.display();
dyncon obj2("cpp"),obj3("python");
obj2.display();
obj3.display();
return 0;
}
```

7. Anonymous Objects.

- It is possible to declare objects without any name.
- These objects are said to be anonymous objects.
- constructors and destructors are called automatically whenever an object is created and destroyed respectively.
- the anonymous objects are used to carry out these operations without object.

```
class noname
{
int x;
public:
noname()
{
cout<<“\n In Default Constructor”;
x=10;
cout<<x;
}
noname(int i)
{
cout<<“\n In Parameterized Constructor”;
x=i;
cout<<x;
}
~noname()
{
cout <<“\n In Destructor”;
}
};
void main()
{
noname();
noname(12);
}
```

Constructor and destructor with static members

- Every object has own set of data members.when a member function is invoked,only one copy of data member of calling object is available to the function.
- Sometimes,it is necessary for all the objects to share fields which are common for all the objects.if member variable is declared as static,only one copy of such member is created for entire class.
- All objects access the same copy of static variable.
- The static member variable can be used to count the number of objects declared for a particular class.