# Constructors and Destructors

Unit III

Chapter 5

# Constructors

- A constructor is a special member function whose task is to initialize the objects of its class.
- It is special because its name is same as the class name.
- The constructor is invoked whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class.

# Constructor - example

```
class add
{
     int m, n ;
   public :
     add (void) ;
     ------
};
add :: add (void)
{
   m = 0; n = 0;
}
```

- When a class contains a constructor, it is guaranteed that an object created by the class will be initialized automatically.

- add a ;
  - Not only creates the object a of type add but also initializes its data members m and n to zero.

# Constructors

continue …

- There is no need to write any statement to invoke the constructor function.
- If a ‘normal’ member function is defined for zero initialization, we would need to invoke this function for each of the objects separately.
- A constructor that accepts no parameters is called the **default constructor**.
- The default constructor for class A is A : : A ( )

# Characteristics of Constructors

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and they cannot return values.

# Characteristics of Constructors

continue …

- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, Constructors can have default arguments.
- Constructors can not be virtual.

# Characteristics of Constructors

continue …

- We can not refer to their addresses.

- They make ‘implicit calls’ to the operators ***new*** and ***delete*** when memory allocation is required.

# Constructors

continue …

- When a constructor is declared for a class, initialization of the class objects becomes mandatory.

# 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.

# Parameterized Constructors

continue …

```
class add
{
     int m, n ;
   public :
     add (int, int) ;
     ------
};

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

- When a constructor is parameterized, we must pass the initial values as arguments to the constructor function when an object is declared.
- Two ways of calling:
  - Explicit
    - add a1 = add(2,3);
  - Implicit
    - add a1(2,3)
    - Shorthand method

```
//Parameterised constructor

class Point
 {
    private :
              float x, y;

    public:

          Point (int a, int b)//constructor definition
          {
             x=a;
             y=b;
          }

          void  display()
          {
               cout<<x<<“    ”<<y <<”\n”;
          }
     };
```

```
void main()
 {
   Point p1(1,1);
   Point p2(5,10);

   p1.display();

   p2.display();

   return 0;

}
```

# Destructors

- A destructor is used to destroy the objects that have been created by a constructor.

- Like constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde.

eg: ~ integer ( ) { }

# Destructors

continue …

- A destructor never takes any argument nor does it return any value.

- It will be invoked implicitly by the compiler upon exit from the program – or block or function as the case may be – to clean up storage that is no longer accessible.

# Destructors

continue …

- It is a good practice to declare destructors in a program since it releases memory space for further use.

- Whenever ***new*** is used to allocate memory in the constructor, we should use ***delete*** to free that memory.

```
//Implementation of destructors

int count=0;
class test
{
  public:
    test()
     {
      count++;
      cout<<"\nConstructor msg : Object number "<<count<<"created...";

       }
      ~test()
       {
        count<<"\nDestructor msg: Object number "<<count<<"destroyed..";
        count--;
       }
};

int main()
{
  cout<<"\n Inside the main block...";
  cout<<"\n creating first object T1...";
  test T1;
   {    //Block 1
    cout<<"\nInside Block 1....";
    cout<<"\nCreating two more objects T2 and T3...";
    test T2,T3;
    cout<<"\nLeaving Block 1...";
   }
 cout<<"\nBack inside the main block...";
}
```

Output of this program:

```
inside the main block
creating first object T1
constructor a: object number1 created
inside Block-1
creating two more object T2 and T3
constructor msg: object number 2 created
constructor msg: object number 3 created
leaving Block-1
Destructor msg: object number 3 destroyed
Destructor msg: object number 2 destroyed
back inside the main block
Destructor msg: object number 1 destroyed
```

Thank You

# Default vs implicit constructor

- A default constructor is a constructor which can be called with no arguments (either defined with an empty parameter list, or with default arguments provided for every parameter).

- If no user-defined constructors of any kind are provided for a class type, the compiler will always declare a default constructor.