Constructor

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

Example 1:

```
class point
{ int m , n ;
  public:
        point ( )      // constructor
        {
          m = 0 ; n = 0 ;
        }
} ;

main ( )
{
 point  X1 ;  //not only creates the object X1 of type point but also initializes it data members m and n to  zero
}
```

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

This is useful because the programmer may forget to initialize the object after creating it. It's more reliable and convenient, especially when there are a great many objects of a given class, to cause each object to initialize itself when it's created. The constructor does this.

Types of Constructor:

A constructor with no parameters is called (*Default Constructor*) and
a constructor that can take arguments is called (*Parameterized Constructor*).

The constructor functions have some special characteristics:

1- They should be declared in public section.
2- They are executed automatically when the objects are created.

3- They do not have return types, not even void and therefore, they cannot return values.

Parameterized Constructors

C++ permits to pass argument to the constructor function when the objects are created.

Example 2 : Rectangle Class with constructor

```
# include < iostream.h >
class Rectangle
{ int length , width ;
  public:
    Rectangle ( int a , int b )      // constructor
   {  length = a  ;
      Width = b  ; }
     int area ( )
     {  return length * width ;  }
 } ;
main ( )
{ Rectangle R (10 , 4 ) ;    // implicit call
  cout << R . area ( ) ;
}
```

The initial values passed as arguments to the constructor function when an object is declared this can be done in two ways (**types of constructor calling**):

1- By calling the constructor explicitly.
2- By calling the constructor implicitly.

The following declaration illustrates the first method:

Rectangle R1 = Rectangle (9 , 2) ; //*explicit* call

This statement creates a Rectangle object R1 and passes the values 9 and 2 to it.

The second method is implemented as follows:

Rectangle R1 (9 , 2) ; // *implicit* call

This method, sometimes called the *shorthand* method, is used very often as it is shorter, looks better and is easy to implement.

Multiple Constructors in a Class

Example 3:

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

```
class AB
{ private: int x , y ;
  public:
          AB( )
          {  }
          AB ( int z , int r )
          {   x = z  ;
              y = r  ;  }
} ;
main ( )
{ AB  m ;    // call constructor1
  AB  n ( 9 , 2 ) ;  // call constructor2
}
```

When more than one constructor function is defined in class the process is called *constructor overloading*, as in the following example. Generally, the process of defining more than one function having the same name is called *function overloading*.

In the above example there is the empty constructor which is do nothing constructor (defined to satisfied (يرضي) the compiler), since there is the parameterized constructor, the empty constructor must be defined otherwise the compiler will show error message.

Example 4: shows *multiple constructors* and *scope resolution operator*.

```
# include < iostream.h>
class Rectangle
{ public: int length , width ;
            Rectangle ( )
             { length = 0 ;
               width = 0 ;  }
             Rectangle ( int x , int y )
             {  length = x ;
                width = y ; }
             int area ( ) ;
             void show ( ) ;
  } ;
int Rectangle :: area ( )
{  return length  *  width  ; }

void Rectangle :: show ( )
 { cout << length ;
```

```
   cout << width ; }

main ( )
{ Rectangle R1 , R2 ( 20 , 8 ) ;
  R1 . length = 35 ;
  R1 . width  =  12 ;
R1 . show ( ) ;
R2 . show ( ) ;
cout << " area  of  R1 = " << R1 . area ( ) ;
cout << " area  of  R2 = " << R2 . area ( ) ;
}
```

H.W : Write a program that uses the above program but all data are private.

Destructor

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

The destructor is a member function whose name is the same as the class name but it preceded by a tilde (**~**) symbol.
For example the destructor for the class *Rectangle* can be defined as:

```
~ Rectangle ( )
  {    }
```

The destructor never takes any argument nor does it return any value.
It will be called implicitly by the compiler upon exit from the program to clean up the storage that is no longer accessible.
Unlike constructors, a class may have at most one destructor.

Example 1:

```
# include < iostream.h >
class ABC
{ public : int x , y ;
  public : ABC ( )
           {  x = 2  ;
              y  = 7  ; }

           ~ ABC ( )
             {  }
 } ;
main ( )
```

```
{ ABC m ;
  cout <<  m . x << endl ;
  cout <<  m . y << endl ;
}
```

Output : 2
7

Example 2 : Employee class whose member data (name , age , salary , dept). Its member functions (input and show). Using constructor and destructor.

```
# include < iostream.h>
const  int  size  =  30 ;

class employee
{ public:  char  name [ size ]  ;
           int  age , salary ;
           char  dept [ size ]  ;

           employee ( )
             {       }

          ~ employee ( )
             {     }

           void input ( )
           { cin >> name ;
             cin >> age ;
             cin >> salary ;
             cin >> dept ;
           }

           void show ( )
           { cout << name << endl ;
             cout << age << endl ;
             cout << salary << endl ;
             cout << dept << endl ;
            }
   } ;

main ( )
{ employee  emp  ;
  emp . input ( ) ;
```