Constructors
&
Destructors

Constructors

- Constructor is a special ***member function*** whose name is same as that of the class in which it is defined and it does not return anything (not even void).
- Constructor's primary task is to ***allocate memory for the object*** and as a secondary task it may ***initialize the data members*** of the object

```
class X{
    int a;
    public:
              X()
                  {    a=5;  }
              void disp()
                  {    cout<<a;  }
};
```

```
main{
    X x1=X();   // Explicit Call
    x1.disp();  // 5

    X x2();     // Implicit Call
    x2.disp();  // 5

    X x3;       // Implicit Call
    x2.disp();  // 5
}
```

- Constructor with no arguments is known as the default constructor
- If the user does not specify the constructor then compiler provides a default constructor into the class.

```
X()
  {
          // no statement inside the constructor
  }
```

It is also called as “Do Nothing Constructor”

Properties of Constructor

1. It should be declared in the public section but we can have private constructors.
2. Constructor is invoked automatically when the object is created.
3. They don’t have any return type, not even void.
4. They can’t be inherited but, the derived class can call the base class constructor.
5. They can have default arguments.
6. Constructors can’t be virtual.
7. We can’t refer to the address of the constructor.
8. An object having a user-defined constructor can’t be made as a member of an union.

Parameterized Constructor

- The constructor taking at least one parameter is called a parameterized constructor.

```
class  X{
            int a;
       public:
            X(int b)
                {   a=b;   }
            void disp()
                {    cout<<a;    }
     } ;
```

```
main{
   X x1=X(5);   // Explicit Call
   x1.disp();      // 5

   X x2(10);   // Implicit Call
   x2.disp();     // 10

   X x3=8;      // Implicit Call
   x3.disp().      // 8
}
```

Parameterized Constructor (Contd..)

```
class  X{
            int a,b;
        public:
            X(int c.int d)
                {    a=c;
                      b=d;
                }
            void disp()
                {     cout<<a<<b;    }
    } ;
```

```
main{
    X x1=X(5,8);   // Explicit Call
    x1.disp();         // 5, 8

    X x2(10,20);   // Implicit Call
    x2.disp();        // 10, 20

    X x3=(4,6);        // Error
    x3.disp().

   X x4=4,6;           // Error
   x3.disp().
}
```

Constructor Overloading

More than one constructors inside a class which can be differentiated by the compiler at the compile time based on the no of args, type of args or sequence of args , is called as Constructor Overloading

```
class  X{
            int a,b;
        public:
            X()   {  a=b=0;  }
            X(int c)   {  a=b=c;  }
            X(int c.int d)
                {    a=c;
                     b=d;
                }
            void disp()
                {     cout<<a<<b;    }
     };
```

```
main{
    X x1(5);
    x1.disp();           // 5, 5

    X x2(10,20);
    x2.disp();          // 10, 20

    X x3;
    x3.disp().          //  0, 0

   x3=x2;
   x3.disp().          // 10, 20
}
```

Copy Constructor

- It is used to copy the content of one object to another object of same class.

```
class  X{
            int a;
       public:
            X(int k) { a=k; }
            // Copy contructor
            X(X &p)   {  a=p.a;  }
            void disp()
                {    cout<<a;    }
     };
```

```
main{
   X x1(5);
   X x2(x1);   //calls the copy const
   x1.disp();       // 5
   x2.disp().       // 5
}
```

- Copy constructor takes a reference type as argument otherwise it leads to an infinite loop.
- Compiler provides a default copy constructor if the programmer has not specified it explicitly.

Default arguments in Constructor

```
class  X{
            int a,b;
       public:
            X(int c,int d=0) {
                        a=c;
                        b=d;
           }
            void disp()
                {    cout<<a<<b;    }
     };
```

```
main{
   X x1(5,7);
   X x2(1);
   x1.disp();      // 5, 7
   x2.disp().      // 1, 0
}
```

Destructor

- It de-allocates the memory of the object which was allocated by a constructor.
- Destructors are called automatically when the object goes out of scope.
- The name of the destructor is same as class name and must be preceded by a tilde mark (~).

```
int i=0;
class  X{
            int a;
       public:
            X() {  cout<<“CON”<<++i }
            ~X(){   cout<<“DES”<<i--; }
     } ;
```

```
main{
   X x1,x2;
}
```

Output:

```
CON1
CON2
DES2
DES1
```

Destructors are called in the reverse order of constructors.

```
int i=0;
class  X{
            int a;
       public:
            X() {  cout<<“CON”<<++i }
            ~X(){   cout<<“DES”<<i--; }
     } ;
```

```
main{
   X x1,x2;
      {
           X x3,x4;
       }
      {
           X x5;
      }
}
```

Output:

```
CON1
CON2
CON3
CON4
DES4
DES3
CON3
DES3
DES2
DES1
```

Properties of Destructor

- Name of the destructor is same as the class name preceded by ~ mark.
- It does not have any return type, not even void.
- It can’t be inherited but the derived class may call the base class destructor.
- We can have only one destructor in a class i.e. Destructor can’t be overloaded as it does not take any argument.
- We can’t refer to the address of the destructor.
- Unlike constructor, it can be virtual.
- Object having a destructor can’t be made as a member of union.