CHAPTER 7: CONSTRUCTORS AND DESTRUCTORS

Definition of Constructors

Constructors are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors iitialize values to object members after storage is allocated to the object

```
class A
{
    int x;
    public:
    A(); //Constructor
};
```

While defining a contructor you must remeber that the name of constructor will be same as the name of the class, and contructors never have return type.

Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution"::" operator.

```
class A
{
    int i;
    public:
    A(); //Constructor declared
    };
    A::A() // Constructor definition
    {
    i=1;
}
```

Types of Constructors

Constructors are of three types :

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor

Default Constructor

Default constructor is the constructor which doesn't take any argument. It has no parameter.

Syntax :

```
class_name ()
{ Constructor Definition }
```

Example :
```
        class Cube
        {
              int side;
              public:
              Cube()
              {
                    side=10;
              }
        };
        int main()
        {
              Cube c;
              cout << c.side;
}
```

Output : 10

In this case, as soon as the object is created the constructor is called which initializes its data members.

A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

```
      class Cube
          {
                int side;
          };
      int main()
      {
           Cube c;
           cout << c.side;
          }
```

Output : 0

In this case, default constructor provided by the compiler will be called which will initialize the object data members to default value, that will be 0 in this case.

Parameterized Constructor

These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

Example :

```
class Cube
{
        int side;
        public:
        Cube(int x)
            {
                 side=x;
            }
};
int main()
{
          Cube c1(10);
          Cube c2(20);
          Cube c3(30);
          cout << c1.side;
          cout << c2.side;
          cout << c3.side;
}
```

OUTPUT : 10 20 30

By using parameterized construcor in above case, we have initialized 3 objects with user defined values. We can have any number of parameters in a constructor.

Copy Constructor

These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object. We will study copy constructors in detail later.

Constructor Overloading

Just like other member functions, constructors can also be overloaded. Infact when you have both default and parameterized constructors defined in your class you are having Overloaded Constructors, one with no parameter and other with parameter.

You can have any number of Constructors in a class that differ in parameter list

```
class Student
{
     int rollno;
     string name;
     public:
     Student(int x)
     {
```

```
                rollno=x;
                name="None";
             }
            Student(int x, string str)
            {
                rollno=x ;
                name=str ;
            }
        };
        int main()
        {
            Student A(10);
            Student B(11,"Ram");
               }
```

In above case we have defined two constructors with different parameters, hence overloading the constructors.

One more important thing, if you define any constructor explicitly, then the compiler will not provide default constructor and you will have to define it yourself.

In the above case if we write Student S; in **main()**, it will lead to a compile time error, because we haven't defined default constructor, and compiler will not provide its default constructor because we have defined other parameterized constructors.

Destructors

Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.
The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a **tilde** ~ sign as prefix to it.

```
class A
{
   public:
   ~A();
};
```

Destructors will never have any arguments.

Example to see how Constructor and Destructor is called

```
class A
{
     A()
      {
```

```
                cout << "Constructor called";
        }
    ~A()
        {
              cout << "Destructor called";
        }
};
int main()
{
    A obj1; // Constructor Called
    int x=1
    if(x)
      {
          A obj2; // Constructor Called
      } // Destructor Called for obj2
} // Destructor called for obj1
```

Implementation of constructors and Destructors

Single Definition for both Default and Parameterized Constructor

In this example we will use **default argument** to have a single definition for both defualt and parameterized constructor.

```
class Dual
{
    int a;
    public:
    Dual(int x=0)
      {
          a=x;
      }
};
int main()
{
    Dual obj1;
    Dual obj2(10);
}
```

Here, in this program, a single Constructor definition will take care for both these object initializations. We don't need separate default and parameterized constructors.