

# Constructor

---

- A constructor is a special member function automatically called when an object is created.
- In C++, the constructor is automatically called when an object is created.
- It is a special class method because it does not have any return type.
- It has the same name as the class itself.
- A constructor initializes the class data members with garbage value if we don't put any value to it explicitly.
- The constructor must be placed in the public section of the class because we want the class to be instantiated anywhere.
- For every object in its lifetime constructor is called only once at the time of creation.

## Example:

```
class class_name{
    int data_member1;
    string data_member2;

    //creating constructor
public:
    class_name(){
        // initialize data members with garbage value
    }
};
```

Here, the function `class_name()` is a constructor of the class 'class\_name'.

Notice that the constructor

- ❖ has the same name as the class,
- ❖ does not have any return type,
- ❖ it is public

- If we do not specify a constructor, the C++ compiler generates a default constructor for an object (which expects no parameters and has an empty body).

### Types of Constructors:

Three types of constructors in C++:

Default          Parameterized          Copy

### Default constructor:-

- A constructor that doesn't take any argument or has no parameters is known as a default constructor.
- In the example above, `class_name()` is a default constructor.

```
class class_name{  
    int data_member1;  
    string data_member2;  
  
    //default constructor  
public:  
    class_name(){  
        // initializing data members with their default values  
        data_member1 = 69;  
        data_member2 = "Coding Ninjas";  
    }  
};
```

Syntax:

Here, the `class_name()` constructor will be called when the object is created.

This sets the `data_member1` variable of the object to 69 and the `data_member2` variable of the object to "Coding Ninjas".

Note:

- If we have not defined a constructor in our class, the C++ compiler will automatically create a default constructor with an empty code and no parameters, which will initialize data members with garbage values.

- When we write our constructor explicitly, the inbuilt constructor will not be available for us.

### Parameterized Constructor:-

- Constructor with parameters.
- The parameterized constructor takes its arguments provided by the programmer.
- These arguments help initialize an object when it is created.
- To create a parameterized constructor, simply add parameters to it the way you would to any other function.
- When defining the constructor's body, use the parameters to initialize the object.
- Using this Constructor, you can provide different values to data members of different objects by passing the appropriate values as arguments.

```
class class_name{  
    int data_member1;  
    string data_member2;  
  
    //parameterized constructor  
public:  
    class_name(int num, string str){  
        // initializing data members with values provided  
        data_member1 = num;  
        data_member2 = str;  
    }  
};
```

Syntax:

- Here, we have created a parameterized constructor class\_name() that has 2 parameters: int num and string str.
- The values contained in these parameters are used to initialize the member variables data\_member1 and data\_member2.

## Copy Constructor:-

- These are a particular type of constructor that takes an object as an argument and copies values of one object's data members into another object.
- We pass the class object into another object of the same class in this constructor.
- Copy means to copy the values of one Object into another Object of Class.
- This is used for Copying the values of a class object into another object of a class, so we call them Copy constructor and for copying the values.
- We have to pass the object's name whose values we want to copy, and when we are using or passing an object to a constructor, we must use the & ampersand or address operator.

```
class class_name{  
    int data_member1;  
    string data_member2;  
  
    //copy constructor  
public:  
    class_name(class_name &obj){  
        // copies data of the obj parameter  
        data_member1 = obj.data_member1;  
        data_member2 = obj.data_member2;  
    }  
};
```

### Syntax:

- In this program, we have used a copy constructor to copy the contents of one object of the class 'class\_name' to another.
- The code of the copy constructor is:

```
class_name(class_name &obj){  
    // copies data of the obj parameter  
    data_member1 = obj.data_member1;  
    data_member2 = obj.data_member2;  
}
```

If we don't define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a memberwise copy between objects.

Example using smartphone class:

```
class smartphone{  
    //Data Members(Properties)  
    string model;  
    int year_of_manufacture;  
    bool _5g_supported;  
  
    public:  
    //default constructor  
    smartphone() {  
        model = "unknown";  
    }  
}
```

```

    year_of_manufacture = 0;
    _5g_supported = false;
}
//parameterized constructor
smartphone(string model_string, int manufacture, bool _5g_){
    //initialising data members
    model = model_string;
    year_of_manufacture = manufacture;
    _5g_supported = _5g_;
}

// copy constructor
smartphone(smartphone &obj){
    // copies data of the obj parameter
    model = obj.model;
    year_of_manufacture = obj.year_of_manufacture;
    _5g_supported = obj._5g_supported;
}
};
int main(){
    //creating objects of smartphone class

    // using default constructor
    smartphone unknown;

    // using parameterized constructor
    smartphone iphone("iphone 11", 2019, false );

    // using copy constructor
    smartphone iphone_2(iphone);
}

```







