**C++ CONSTRUCTOR AND DESTRUCTOR**

Constructor and Destructor are the **special member functions** of the class that are **created by the C++ compiler** or can be **defined by the user**.

Constructor is used to **initialize the object of the class** while **destructor is called by the compiler when the object is destroyed**.

A constructor is a special member function of a class and **shares the same name as of class**, which means the **constructor and class have the same name**.

Constructor is **called** by the compiler whenever the **object of the class is created**, it allocates the **memory to the object** and **initializes class data members by default values or values passed by the user while creating an object**.

Constructors **don’t have any return type** because their work is to just create and initialize an object.

The basic syntax of the constructor is given below:

**class** **class\_name**{

**private**:

*// private members*

**public**:

*// declaring constructor*

class\_name({parameters})

{

*// constructor body*

}

};

In the above syntax, we can see the class has the name class\_name and the constructor have also the same name. A constructor can have any number of parameters as per requirements. Also, there is no return type or return value of the constructor.  
Note: In the above syntax we have declared the constructor as a public member but we can declare it private also

**How many types of Constructors are present in C++?**

There are four types of constructors in c++

* Default constructor
* Parameterized constructor
* Copy Constructor
* Dynamic Constructor

**Default constructor**

Default constructor is also known as a **zero-argument constructor**, as **it doesn’t take any parameter**.

It can be **defined by the user** if not then the **compiler creates it on his own**.

Default constructor always **initializes data members of the class with the same value** they were defined.



**Example**

**#include <iostream>**

**using** **namespace** std;

**class** **Person**{

*// declaring private class data members*

**private**:

string name;

**int** age;

**public**:

*// declaring constructor*

Person()

{

cout<<"Default constructor is called"<<endl;

name = "student";

age = 12;

}

*// display function to print the class data members value*

**void** **display**()

{

cout<<"Name of current object: "<<name<<endl;

cout<<"Age of current object: "<<age<<endl;

}

};

**int** **main**()

{

*// creating object of class using default constructor*

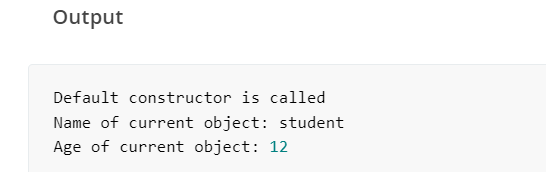
Person obj;

*// printing class data members*

obj.display();

**return** 0;

}



 We have created a class with **two data members**. Declared a default constructor which **always initializes objects of a class with the same name and age**. In the main function, we have **created an object of the class and printed its data member values** by using the display function.

### Parameterized Constructor

Parameterized constructor is used to initialize data members with the values provided by the user. This constructor is basically the upgraded version of the default constructor. We can define more than one parameterized constructor according to the need of the user, but we have to follow the rules of the function overloading, like a different set of arguments must be there for each constructor.



**#include <iostream>**

**using** **namespace** std;

**class** **Person**{

*// declaring private class data members*

**private**:

string name;

**int** age;

**public**:

*// declaring parameterized constructor of three different types*

Person(string person\_name)

{

cout<<"Constructor to set name is called"<<endl;

name = person\_name;

age = 12;

}

Person(**int** person\_age)

{

cout<<"Constructor to set age is called"<<endl;

name = "Student";

age = person\_age;

}

Person(string person\_name, **int** person\_age)

{

cout<<"Constructor for both name and age is called"<<endl;

name = person\_name;

age = person\_age;

}

*// display function to print the class data members value*

**void** **display**()

{

cout<<"Name of current object: "<<name<<endl;

cout<<"Age of current object: "<<age<<endl;

cout<<endl;

}

};

**int** **main**()

{

*// creating objects of class using parameterized constructor*

Person **obj1**("First person");

*// printing class data members for first object*

obj1.display();

Person **obj2**(25);

*// printing class data members for second object*

obj2.display();

Person **obj3**("Second person",15);

*// printing class data members for third object*

obj3.display();

**return** 0;

}

**Output**

Constructor to set name is called

Name of current object: First person

Age of current object: 12

Constructor to set age is called

Name of current object: Student

Age of current object: 25

Constructor **for** both name **and** age is called

Name of current object: Second person

Age of current object: 15

In the above code, we have created **three types of the parametric constructor**, one for initialization of **name only**, second to initialization of **age only**, and third to initialize both **name and age**. In the main function, we have created three different types of objects and initialized them in different ways, and printed values for each of them.

**Copy Constructor**

If we have **an object of a class** and we want to create its copy in a new declared object of the same class, then a copy constructor is used.

The compiler provides each class a default copy constructor and users can define it also. It takes a single argument which is an object of the same class.

**Syntax**

**class** **class\_name**{

**private**:

*// private members*

**public**:

*// declaring copy constructor*

class\_name(**const** class\_name& obj)

{

*// constructor body*

}

};

In the above syntax, we created a copy constructor which takes an **object of the same class as a parameter** but it is declared constant and passed as a reference because when an argument is passed as a function parameter it creates a copy for it, to create that copy compiler will again call the copy constructor, means it will call the same function and for that call again there will be a call to create copy which will take this process in neverending recursion of creating copies. To prevent such conditions we pass it as a reference.

**Code to understand the working of the copy constructor**

**#include <iostream>**

**using** **namespace** std;

**class** **Person**{

*// declaring private class data members*

**private**:

string name;

**int** age;

**public**:

Person(string person\_name, **int** person\_age)

{

cout<<"Constructor for both name and age is called"<<endl;

name = person\_name;

age = person\_age;

}

Person(**const** Person& obj)

{

cout<<"Copy constructor is called"<<endl;

name = obj.name;

age = obj.age;

}

*// display function to print the class data members value*

**void** **display**()

{

cout<<"Name of current object: "<<name<<endl;

cout<<"Age of current object: "<<age<<endl;

cout<<endl;

}

};

**int** **main**()

{

*// creating objects of class using parameterized constructor*

Person **obj1**("First person",25);

*// printing class data members for first object*

obj1.display();

*// creating copy of the obj1*

Person **obj2**(obj1);

*// printing class data members for second object*

obj2.display();

**return** 0;

}

**Output**

Constructor **for** both name **and** age is called

Name of current object: First person

Age of current object: 25

Copy constructor is called

Name of current object: First person

Age of current object: 25

In the above code, we have **created a class** and defined two types of constructors in it, the **first is a parameterized constructor** and another is a **copy constructor**. Parameterized constructor is used to **create an object** then by using the copy constructor we create a copy of it and stored it in another object.

**Dynamic Constructor**

When memory is **allocated dynamically** to the data members at the runtime using **a new operator**, the constructor is known as the dynamic constructor.

This constructor is similar to the default or parameterized constructor; the only difference is it uses a **new operator to allocate the memory.**

**Syntax**

**class** **class\_name**{

**private**:

*// private members*

**public**:

*// declaring dynamic constructor*

class\_name({parameters})

{

*// constructor body where data members are initialized using new operator*

}

};

**Code to understand the working of the dynamic constructor**

**#include <iostream>**

**using** **namespace** std;

**class** **Person**{

*// declaring private class data members*

**private**:

**int**\* age;

**public**:

Person(**int**\* person\_age)

{

cout<<"Constructor for age is called"<<endl;

*// allocating memory*

age = **new** **int**;

age = person\_age;

}

*// display function to print the class data members value*

**void** **display**()

{

cout<<"Age of current object: "<<\*age<<endl;

cout<<endl;

}

};

**int** **main**()

{

*// creating objects of class using parameterized constructor*

**int** age = 25;

Person **obj1**(&age);

*// printing class data members for first object*

obj1.display();

**return** 0;

}

**Output**

Constructor **for** age is called

Age of current object: 25

In the above code, we have created a class with a dynamic constructor. In the main function, we have created an object and initialized it using a dynamic constructor, where we have given memory dynamically using a new operator.

**What is Destructor in C++?**

Destructor is just the opposite function of the constructor. A destructor is called by the compiler when the object is destroyed and its main function is to deallocate the memory of the object.

The object may be destroyed when the program ends, or local objects of the function get out of scope when the function ends or in any other case.  
Destructor has the same as of the class with prefix tilde(~) operator and it cannot be overloaded as the constructor. Destructors take no argument and have no return type and return value.

**What is the Basic Syntax of the Destructor?**

**The basic syntax of the Destructor is given below**

**class** **class\_name**{

**private**:

*// private members*

**public**:

*// declaring destructor*

~class\_name()

{

*// destructor body*

}

};

In the above syntax, we can see the class has the name class\_name and the destructor also has the same name, in addition there is a tilde(~). Also, there is no return type and return value of the destructor.  
Note: In the above syntax we have declared the destructor as a public member but we can declare it private also.

**How Constructor and Destructor are called when the object is Created and Destroyed**

As constructor is the first function called by the compiler when an object is created and the destructor is the last class member called by the compiler for an object. If the constructor and destructor are not declared by the user, the compiler defines the default constructor and destructor of a class object.  
Let’s see a code to get the proper idea of how constructor and destructor are called:  
First, we will create a class with single parametrized constructors and a destructor. Both of them contain print statements to give an idea of when they are called.

**#include <iostream>**

**using** **namespace** std;

**class** **class\_name**{

*// declaring private class data members*

**private**:

**int** a,b;

**public**:

*// declaring Constructor*

class\_name(**int** aa, **int** bb)

{

cout<<"Constructor is called"<<endl;

a = aa;

b = bb;

cout<<"Value of a: "<<a<<endl;

cout<<"Value of b: "<<b<<endl;

cout<<endl;

}

*// declaring destructor*

~class\_name()

{

cout<<"Destructor is called"<<endl;

cout<<"Value of a: "<<a<<endl;

cout<<"Value of b: "<<b<<endl;

}

};

**int** **main**()

{

*// creating objects of class using parameterized constructor*

class\_name **obj**(5,6);

**return** 0;

}

**Output**

Constructor is called

Value of a: 5

Value of b: 6

Destructor is called

Value of a: 5

Value of b: 6

In the above code, we have created a class with constructor and destructor. In the main function, an object uses a parametric constructor, and when the program ends the destructor is automatically called by the compiler and we get the values of our variables.