What is Constructor in C++?
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.

What is the basic syntax of Constructor in C++?
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 (we will discuss that later in this article).

Important Points about Constructors
Access specifiers
Constructors can be defined as public, protected, or private as per the requirements. By default or default constructors, which are created by the compiler are declared as the public. If the constructor is created as private, then we are not able to create its object.
When there is no requirement of the object of a class (in a situation when all class members are static) we can define its constructor as private.
Usually, constructors are defined as public because constructors are used to create an object and initialize the class data members for the object. An object is always created from outside of class, which justifies making constructors public.

Inheritance

As a derived class can access all the public properties of the base class, it can call its constructor also if it is not declared as private. Also, the constructor's address cannot be referenced.

Virtual
Constructor in C++ cannot be declared as virtual because when we declare any function of a class as a virtual function, at compile time the compiler creates a virtual table to store the address of each function that is declared as virtual. Also, it creates a data member of the class virtual pointer to points towards the virtual table. But as we have discussed we cannot refer to the address of the constructor, which means we are not able to declare the constructor as virtual.

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.

Syntax:
```
class class_name{
  private:
  // private members

  public:

  // declaring default constructor
  class_name()
  {
    // constructor body
  }

};
```

//Code to show the working of default constructor
```
#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;
}
```

Output

```
Default constructor is called
Name of current object: student
Age of current object: 12
```

In the above code, 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.

Syntax:

```
class class_name{
  private:
  // private members

  public:

  // declaring parameterized constructor
  class_name(parameter1, parameter2,...)
  {
    // constructor body
  }

};
```

Code to understand the working of the parameterized 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.

Important Points about the Destructor

- Destructor are the last member function called for an object and they are called by the compiler itself.
- If the destructor is not created by the user then compile creates or declares it by itself.
- A Destructor can be declared in any section of the class, as it is called by the compiler so nothing to worry about.
- As Destructor is the last function to be called, it should be better to declare it at the end of the class to increase the readability of the code.
- Destructor is just the opposite of the constructor as the constructor is called at the time of the creation of the object and allocates the memory to the object, on the other side the destructor is called at the time of the destruction of the object and deallocates the memory.

How Constructor and Destructor are called when the object is Created and Destroyed
A 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 parameterized 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.

Difference Between Constructor and Destructor in C++

Constructor:

A constructor is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. It can either accept the arguments or not. It is used to allocate the memory to an object of the class. It is called whenever an instance of the class is created. It can be defined manually with arguments or without arguments. There can be many constructors in a class. It can be overloaded but it can not be inherited or virtual. There is a concept of copy constructor which is used to initialize an object from another object.

Syntax:

```
  ClassName()
   {
     //Constructor's Body
```

```
  }
```

Destructor:
Like a constructor, Destructor is also a member function of a class that has the same name as the class name preceded by a tilde(~) operator. It helps to deallocate the memory of an object. It is called while the object of the class is freed or deleted. In a class, there is always a single destructor without any parameters so it can't be overloaded. It is always called in the reverse order of the constructor. if a class is inherited by another class and both the classes have a destructor then the destructor of the child class is called first, followed by the destructor of the parent or base class.
Syntax:

```
 ~ClassName()
  {
     //Destructor's Body
  }
```

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be Private.

Example/Implementation of Constructor and Destructor:

```
#include <iostream>
using namespace std;
class Z
{
public:
   // constructor
   Z()
   {
      cout<<"Constructor called"<<endl;
   }

   // destructor
   ~Z()
   {
      cout<<"Destructor called"<<endl;
   }
};

int main()
{
   Z z1;   // Constructor Called
   int a = 1;
   if(a==1)
```

```
    {
        Z z2;  // Constructor Called
    }  // Destructor Called for z2
} //  Destructor called for z1
```

Output:

```
Constructor called
Constructor called
Destructor called
Destructor called
```

Difference between Constructor and Destructor in C++ :

S. No.	Constructor	Destructor
1.	Constructor helps to initialize the object of a class.	Whereas a destructor is used to destroy the instances.
2.	It is declared as **className(arguments if any){Constructor's Body }**.	Whereas it is declared as **~ className(no arguments){ }**.
3.	Constructor can either accept arguments or not.	While it can't have any arguments.
4.	A constructor is called when an instance or object of a class is created.	It is called while an object of the class is freed or deleted.
5.	Constructor is used to allocate the memory to an instance or object.	While it is used to deallocate the memory of an object of a class.
6.	Constructor can be overloaded.	While it can't be overloaded.
7.	The constructor's name is the same as the class name.	Here, its name is also the same as the class name preceded by the tiled (~) operator.
8.	In a class, there can be multiple constructors.	While in a class, there is always a single destructor.
9.	There is a concept of copy constructor which is used to initialize an object from another object.	While here, there is no copy destructor concept.
10.	They are often called in successive order.	They are often called in reverse order of constructor.