Constructors in C++

What is constructor?

A constructor is a special type of member function of a class which initializes objects of a class.

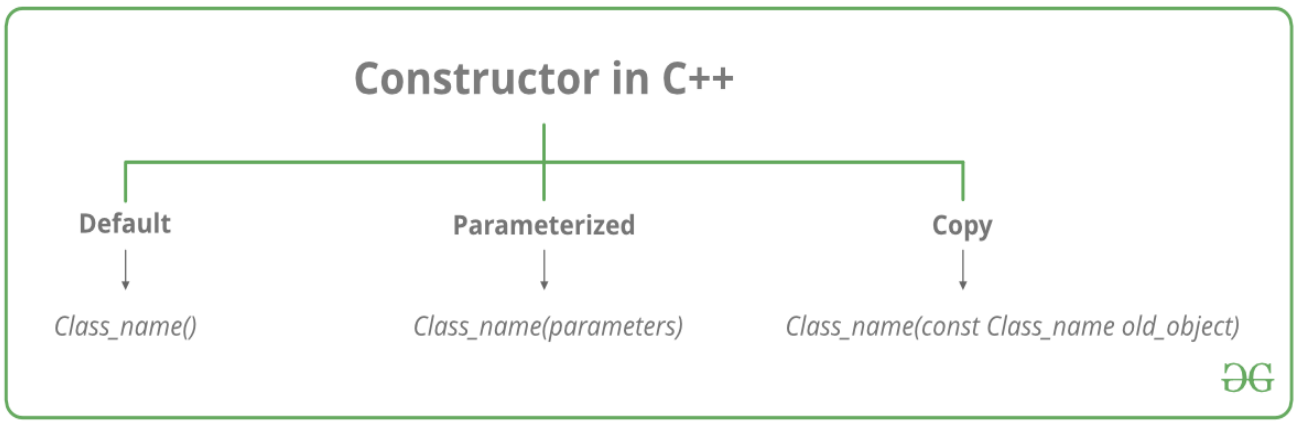
In C++, Constructor is automatically called when object(instance of class) create.

It is special member function of the class because it does not have any return type.

How constructors are different from a normal member function?

A constructor is different from normal functions in following ways:

- Constructor has same name as the class itself
- Constructors don't have return type
- A constructor is automatically called when an object is created.
- It must be placed in public section of class.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).



Syntax:

Class class_name

{

Public:

//variable declaration

Classname() //constructor declaration

{

//body of constructor

}

}

Types of Constructors

1. **Default Constructor:**

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

```
// Cpp program to illustrate the
// concept of Constructors
#include <iostream.h>
#include<conio.h>

class construct
{
public:
   int a, b;

   // Default Constructor
   construct()
   {
      a = 10;
      b = 20;
   }
};

int main()
{
   // Default constructor called automatically
   // when the object is created
   construct c;
   cout << "a: " << c.a << endl
        << "b: " << c.b;
   return 1;
}
```

Output:

```
a: 10
b: 20
```

Note: Even if we do not define any constructor explicitly, the compiler will automatically provide a default constructor implicitly.

2. **Parameterized Constructors:**
   It is possible to pass arguments to constructors.
   Typically, 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 you define the constructor's body, use the parameters to initialize the object.

```
// Cpp program to illustrate the
// concept of Constructors
#include <iostream.h>
#include<conio.h>

class construct
{
public:
   int a, b;

   // Default Constructor
   construct(int x,int y)
   {
      a = x;
      b = y;
   }
};

void  main()
{
   // Default constructor called automatically
   // when the object is created
   construct c(10,20);
   cout << "a: " << a << endl
        << "b: " << b;
   Getch();
}
```

Output:

```
a= 10
b= 20
```

When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly.

```
Example e = Example(0, 50); // Explicit call

Example e(0, 50);           // Implicit call
```

-
-
- **Uses of Parameterized constructor:**
    1. It is used to initialize the various data elements of different objects with different values when they are created.
    2. It is used to overload constructors.
- **Can we have more than one constructor in a class?**
  Yes, It is called Constructor Overloading.

C++ Copy Constructor

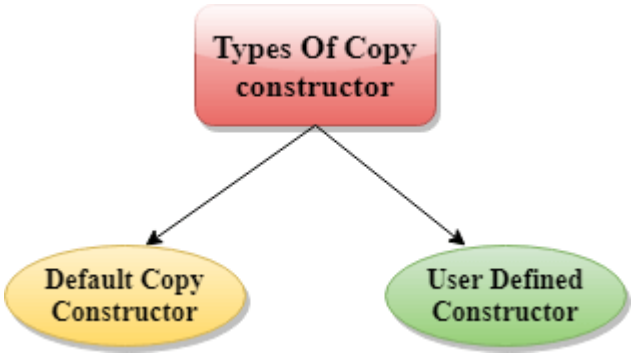
A Copy constructor is an **overloaded** constructor used to declare and initialize an object from another object.

Copy Constructors is a type of constructor which is used to create a copy of an already existing object of a class type.

It is usually of the form **X (X&)**, where X is the class name. The compiler provides a default Copy Constructor to all the classes.

Copy Constructor is of two types:

- **Default Copy constructor:** The compiler defines the default copy constructor. If the user defines no copy constructor, compiler supplies its constructor.
- **User Defined constructor:** The programmer defines the user-defined constructor.



Syntax Of User-defined Copy Constructor:

```
Class_name(const class_name &old_object)

{

.....

}
```

As it is used to create an object, hence it is called a constructor. And, it creates a new object, which is exact copy of the existing copy, hence it is called **copy constructor**.

Consider the following situation:

```
class A
{
   A(A &x) //  copy constructor.
  {
     // copyconstructor.
  }
}
```

In the above case, **copy constructor can be called in the following ways:**

- **A a2(a1);**
- **A a2 = a1;**

} **a1 initialises the a2 object.**

Let's see a simple example of the copy constructor.

// **program of the copy constructor.**

```
#include <iostream.h>
#include<conio.h>
class A
{
  public:
   int x;
   A(int a)              // parameterized constructor.
   {
    x=a;
   }
   A(A &i)               // copy constructor
```

```
    {
        x = i.x;
    }
};
void main()
{
  A a1(20);            // Calling the parameterized constructor.
 A a2(a1);             //  Calling the copy constructor.
 cout<<a2.x;
  getch();
}
```

When Copy Constructor is called

Copy Constructor is called in the following scenarios:

- When we initialize the object with another existing object of the same class type. For example, Student s1 = s2, where Student is the class.
- When the object of the same class type is passed by value as an argument.
- When the function returns the object of the same class type by value.

C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes.

It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

Syntax of Destructor

```
~class_name()
{
   //Some code
}
```

Similar to constructor, the destructor name should exactly match with the class name. A destructor declaration should always begin with the tilde(~) symbol as shown in the syntax above.

When does the destructor get called?

A destructor is **automatically called** when:
1) The program finished execution.
2) When a scope (the { } parenthesis) containing local variable ends.
3) When you call the delete operator.

Note: C++ destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

C++ Constructor and Destructor Example

```
#include <iostream.h>

#include <conio.h>

class Employee

 {

   public:

        Employee()

        {

           cout<<"Constructor Invoked"<<endl;

        }

        ~Employee()

        {

           cout<<"Destructor Invoked"<<endl;

        }

};

void main()

{

    Employee e1; //creating an object of Employee
```

```
    Employee e2; //creating an object of Employee
    getch();
}
```