**Constructors:**

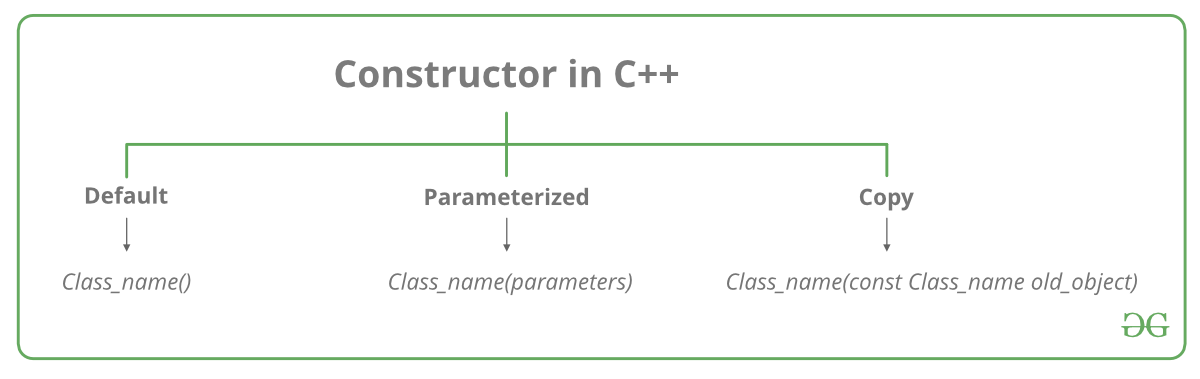
**What is constructor?**

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).



**Types of Constructors:**

1.[**Default Constructors:**](https://www.geeksforgeeks.org/c-internals-default-constructors-set-1/)

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

**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.

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](https://www.geeksforgeeks.org/constructor-overloading-c/).

**3.Copy Constructor:**

**What is a copy constructor?**  
A copy constructor is a member function that initializes an object using another object of the same class. A copy constructor has the following

**general function prototype:**

**ClassName (const ClassName &old\_obj);**

**When is** **copy constructor called?**   
In C++, a Copy Constructor may be called in the following cases:   
1. When an object of the class is returned by value.   
2. When an object of the class is passed (to a function) by value as an argument.   
3. When an object is constructed based on another object of the same class.   
4. When the compiler generates a temporary object.  
It is, however, not guaranteed that a copy constructor will be called in all these cases, because the C++ Standard allows the compiler to optimize the copy away in certain cases, one example is the [return value optimization (sometimes referred to as RVO)](http://en.wikipedia.org/wiki/Return_value_optimization).

# **Constructor Overloading in C++**

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading and is quite similar to [function overloading](https://www.geeksforgeeks.org/function-overloading-c/). 

* Overloaded constructors essentially have the same name (exact name of the class) and differ by number and type of arguments.
* A constructor is called depending upon the number and type of arguments passed.
* While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

# **Destructors in C++**

**What is destructor?**   
Destructor is an instance member function which is invoked automatically whenever an object is going to destroy. Means, a destructor is the last function that is going to be called before an object is destroyed.

The thing to be noted is that destructor doesn’t destroys an object.

Destructors are usually **used to deallocate memory and do other cleanup for a class object** and its class members when the object is destroyed. A destructor is called for a class object when that object passes out of scope or is explicitly deleted.

**Syntax: ~**constructor-name();

**Properties of Destructor:**

* Destructor function is automatically invoked when the objects are destroyed.
* It cannot be declared static or const.
* The destructor does not have arguments.
* It has no return type not even void.
* An object of a class with a Destructor cannot become a member of the union.
* A destructor should be declared in the public section of the class.
* The programmer cannot access the address of destructor.

**When is destructor called?**   
A destructor function is called automatically when the object goes out of scope:   
(1) the function ends   
(2) the program ends   
(3) a block containing local variables ends   
(4) a delete operator is called

**How destructors are different from a normal member function?**   
Destructors have same name as the class preceded by a tilde (~)   
Destructors don’t take any argument and don’t return anything

**Nesting of members:**

#### Code as described/written in the video

// OOPs - Classes and objects

// C++ --> initially called --> C with classes by stroustroup

// class --> extension of structures (in C)

// structures had limitations

// - members are public

// - No methods

// classes --> structures + more

// classes --> can have methods and properties

// classes --> can make few members as private & few as public

// structures in C++ are typedefed

// you can declare objects along with the class declarion like this:

/\* class Employee{

// Class definition

} harry, rohan, lovish; \*/

// harry.salary = 8 makes no sense if salary is private

// Nesting of member functions

#include <iostream>

#include <string>

using namespace std;

class binary

{

private:

string s;

void chk\_bin(void);

public:

void read(void);

void ones\_compliment(void);

void display(void);

};

void binary::read(void)

{

cout << "Enter a binary number" << endl;

cin >> s;

}

void binary::chk\_bin(void)

{

for (int i = 0; i < s.length(); i++)

{

if (s.at(i) != '0' && s.at(i) != '1')

{

cout << "Incorrect binary format" << endl;

exit(0);

}

}

}

void binary::ones\_compliment(void)

{

chk\_bin();

for (int i = 0; i < s.length(); i++)

{

if (s.at(i) == '0')

{

s.at(i) = '1';

}

else

{

s.at(i) = '0';

}

}

}

void binary::display(void)

{

cout<<"Displaying your binary number"<<endl;

for (int i = 0; i < s.length(); i++)

{

cout << s.at(i);

}

cout<<endl;

}

int main()

{

binary b;

b.read();

// b.chk\_bin();

b.display();

b.ones\_compliment();

b.display();

return 0;

}