**CONSTRUCTOR**

* Constructor is a member function of a class
* Implicitly invoked when object is created.
* It has no return type so it can’t use return keyword.
* It is used in construction of the object i.e. used to initialize the object.
* Constructor is needed as suppose we create the object of the class and don’t have constructor in the class, class has a data member ‘x’, now if we don’t initialize x with any value, it will be given ‘garbage value’ and work in done on that garbage value.
* It must be an instance member function, it can’t be static.
* **TYPES OF CONSTRUCTOR**
* **Default constructor**: - A constructor that takes no arguments are called default constructors. Default constructors are of two types:-

1. Implicit default constructor: when user doesn’t make any constructor in class, compiler itself makes a default constructor for the class.
2. **Explicit default constructor**: It is the user defines a default constructor in class.

* **Parameterized constructor** is the one that accepts one or more argument.
* **Copy constructor** is the one that takes object of class as the argument and copies value of members of one object to the values of member of another object.
* **Dynamic constructor** are those constructors in which memory for data members is allocated dynamically.(see book for example)
* **Important:**

1. When user doesn’t make any constructor in the class, two constructors are made implicitly by the compiler: one is default constructor and other is copy constructor.
2. When user make any type of constructor apart from copy constructor then compiler implicitly makes copy constructor by itself
3. When copy constructor is made by the user then compiler doesn’t make any of the constructor implicitly.

* Why object if passed by reference not by value?

**ANS:** In copy constructor, object is passed by reference not by value because if we pass by value like base (base b) then Copy constructor will call itself to copy actual parameters to formal parameters and then that parameters will call copy Constructors to again make a copy of formal parameters so making an endless chain but when we pass by reference, the receiving Variable becomes alias (duplicate) of supplied object. Hence no need to call copy constructor to copy actual arguments to formal arguments.

* **DESTRUCTOR** is also a member function which is automatically invoked when object goes out of scope to destroy the object.

>>it is used to deallocate the dynamic memory.

>> ~base() {}

>> invoked in reverse order as that of constructor.

>> A class has only one destructor and it also doesn’t take any argument so,it cant be overloaded.

>>**constructor and destructor can’t be inherited.** A logical reason is that constructor is used in creation of particular class object i.e initialization of data-members of object but suppose if constructors can be inherited then it should also initialize the members of class inheriting it which is quite ambiguous. Same with destructors

* **If the programmers doesn’t define destructor then compiler automatically declare destructor as inline public member function of its class**
* C++ allows constructors and destructors to be declared in private section but then it cannot be called implicitly. They have to be called through public member function.
* We can call a destructor from a constructor

Class x {

X () {

X :: ~ x();

}

} // destructor is being called using class name

* We can call constructor from destructor

Class x {

~X () {

x();

}

} /\* constructor is being called without using class name ,however it may result in recursive call of constructor-destructor till stack overflows and abnormal program termination occurs \*/