Constructor

* Constructor is a special member function, that initializes the object (instance) of its class
* Constructor has the same name as the class and has no return value
* The constructor is invoked whenever an object of its associated class is created.

(Eg) (1) // Default constructor.

```
class integer
 {
  int m,n;
  public:
    integer( )    // Constructor (default)
     {
      m=0;
      n=0;
     }
  void display( )
     {
      cout << m << n ; //prints 0 0
     }
 };
 void main( )
  {
   integer int1 ;  // The object int1 on creation is initialized
   int1.display( );  automatically (ie) its data members m
  }                  and n are initialized to zero.
```

* A constructor that accepts no parameter is called the default constructor.
* If no default constructor is defined, the compiler supplies default constructor.

Note: a constructor should be declared in public section.

(ii) Parameterized constructor

* Constructor that can take arguments are called parameterized constructor.

(Eg)
```
// Parameterized constructor.
Class integer
{
 int m,n;
 public:
   integer (int x, int y)  // parameterized constructor
    {
     m=x; n=y;
    }
   void display()
    {
     cout<<m<<n;  // prints 50 60
    }
};
Void main()
{
 integer int1(50,60); // implicit call => pass initial values as
                      arguments to constructor fn. when
                      object is created.
}
```

Note:

```
integer int1 = integer(50,60); // explicit call.
```

(iii) Copy constructor

* A copy constructor is a member function which initializes an object using another object of the same class.

Syntax:

```
classname ( classname &oldobj)
  {
  }
```

(Eg) // Copy constructor

```
#include <iostream.h>

Class code
{
 int id;
public:
  code()   // do-nothing implicit constructor → defined to satisfy compiler.
   {
   }
  code(int a)   // parameterized constructor.
   {
    id = a;
   }
  code ( code &x) // copy constructor ⇒ reference variable is used as arg. to copy constructor.
   {                 Cannot pass arg. by value to copy constructor.
    id = x.id;
   }
   void display()
    {
     cout<<id;
    }
  };
void main()
 {
  code A(100);
  code B(A); // copy constructor.
  code C = A; // copy constructor.
  cout << "Id of A:";
   A.display();
  cout << "Id of B:";
   B.display();
  cout << "Id of C:";
   C.display();
 }
```

O/P:

```
Id of A : 100
Id of B : 100
Id of C : 100.
```

Note: C++ compiler has an implicit constructor that creates objects. However, it is not defined in class.

* Multiple constructors in a class (overloaded constructor).

(Eg)
```
class integer
{
int m,n;
public:
   integer()          // default constructor
    {
     m=0; n=0;
    }
   integer(int a, int b) // parameterized constructor.
    {
     m=a; n=b;
    }
   integer (integer &i)   // copy constructor.
    {
     m = i.m; n= i.n;
    }
   void display()
    {
     cout << m << n;
    }
};
void main()
 {
  integer i1; => invokes default constructor and set m & n as zero.
  i1.display(); // prints 0 0
  integer i2(20,40); => invokes parameterized constructor.
  i2.display(); // prints 20 40
  integer i3(i2); => copies values of i2 into i3. Also,
   i3.display(); // prints 20 40
 }
```

integer i2 = i3; // valid
i2 = i3; // But it represents overloaded assignment operator

Destructor

* A destructor is used to destroy the objects, that have been created by a constructor.
* Destructor is a member function, whose name is same as class name, but is preceded by a tilde.

(Eg) Destructor for the class integer is

```
~integer()
 {
 }
```

* A destructor never takes any argument.
* Does not return any value.
* Invoked implicitly by compiler upon exit from program (or) block (or) function → to clean up storage.
* Good practice to declare destructors in a program - to release memory space for future use.

(Eg)
```
#include<iostream.h>
int count=0;
class alpha
{
public:
 alpha() // constructor
 {
  count++;
  cout<<"\nNo. of the object created:"<<count;
 }
 ~alpha() //destructor
 {
  cout<<"\n No. of the object destroyed"<<count;
  count--;
 }
};
void main()
{
cout<<"\n Enter main";
alpha A1, A2;
{
cout<<"\n Enter Block 1";
 alpha A3;
}
{
cout<<"\n Enter Block2";
 alpha A4;
}
cout<<"\n Re-enter main";
}
```

O/P:

```
Enter main
No. of the object created : 1
No. of the obj. created : 2
Enter Block 1
No. of the obj. created : 3
No. of the obj. destroyed 3
Enter Block 2
No. of the obj. created : 3
No. of the obj. destroyed 3
Re-enter main
No. of the obj. destroyed : 2.
No. of the obj. destroyed : 1
```