The Class Constructor

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

We can declare constructor in two form:
1. Simple Constructor inside the class

```
class Line {
   public:
      void get();
      Void display();
      Line();  // This is the constructor
   };
```

2. Declaring constructor outside the class

```
class A
{
int i; public:
A(); //Constructor declared
};
A::A() // Constructor definition
{
i=1;
}
```

Types of Constructor

Constructors are of five types:

1. **Default Constructor**
2. **Parametrized Constructor**
3. **Copy Constructor**
4. **Dynamic Constructor**
5. **Overloading Constructor**

Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

```
class Employee
 {
   public:
        Employee()
        {
            cout<<"Default Constructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```

C++ Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects

```
class Employee {
   public:
       int id;//data member (also instance variable)
       string name;//data member(also instance variable)
       float salary;
       Employee(int i, string n, float s)
        {  id = i;
            name = n;
            salary = s;
        }
       void display()
        {
            cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};
int main(void) {
    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
    Employee e2=Employee(102, "Nakul", 59000);
    e1.display();
    e2.display();
    return 0;
}
```

Copy Constructor:

A copy constructor is a member function which initializes an object using another object of the same class. Detailed article on Copy Constructor.

```
class point {
private:
  double x, y;

public:
  // Non-default Constructor & default Constructor
  point (double px, double py) {
    x = px, y = py;
  }
};

int main(void) {
  // Define an array of size 10 & of type point
  // This line will cause error
  point a[10];

  // Remove above line and program will compile without error
  point b = point(5, 6);
}
```

Dynamic Constructor

When allocation of memory is done dynamically using dynamic memory allocator new in a constructor, it is known as dynamic constructor. By using this, we can dynamically initialize the objects.

```
class geeks {
    const char* p;
                                                            Doubt
public:
    // default constructor
    geeks()
    {

        // allocating memory at run time
        p = new char[6];
        p = "geeks";
    }

    void display()
    {
        cout << p << endl;
    }
};

int main()
{
    geeks obj = new geeks();
    obj.display();
}
```

Constructor Overloading

The constructor overloading has few important concepts.

Overloaded constructors must have the same name and different number of arguments

The constructor is called based on the number and types of the arguments are passed.

We have to pass the argument while creating objects, otherwise the constructor cannot understand which constructor will be called.

```
class Rect{
   private:
   int area;
   public:
   Rect(){
      area = 0;
   }
   Rect(int a, int b){
      area = a * b;
   }
   void display(){
      cout << "The area is: " << area << endl;
   }
};

main(){
   Rect r1;
   Rect r2(2, 6);
   r1.display();
   r2.display();
}
```

Destructors in C++

Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.

The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a tilde ~ sign as prefix to it.

```
class A
{
    public:
    // defining destructor for class
    ~A()
    {
        // statement
    }
};
```

```
class A
{
    // constructor
    A()
    {
        cout << "Constructor called";
    }

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

int main()
{
    A obj1;   // Constructor Called
    int x = 1
    if(x)
    {
        A obj2;  // Constructor Called
    }   // Destructor Called for obj2
} //  Destructor called for obj1
```

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

Destructor rules

1) Name should begin with tilde sign(~) and must match class name.

2) There cannot be more than one destructor in a class.

3) Unlike constructors that can have parameters, destructors do not allow any parameter.

4) They do not have any return type, just like constructors.

5) When you do not specify any destructor in a class, compiler generates a default destructor and inserts it into your code.

10. List down the characteristics of destructors.

- They are invoked automatically when the objects are destroyed.
- They obey the usual access rules that other member functions do.
- They cannot be inherited.
- No argument can be provided to a destructor, neither does it return any value.
- They cannot be virtual.
- We cannot refer to their addresses.
- They cannot be static.
- If there is no destructor in a class, a default destructor is generated by the compiler.