**Constructors and Destructors**

Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.

Constructor does not have a return value, hence they do not have a return type.

The prototype of Constructors is as follows:

<class-name> (list-of-parameters);

Constructors can be defined inside or outside the class declaration:-

The syntax for defining the constructor within the class:

<class-name> (list-of-parameters) { // constructor definition }

The syntax for defining the constructor outside the class:

<class-name>: :<class-name> (list-of-parameters){ // constructor definition}

```
#include <iostream>
using namespace std;

class student {
int rno;
char name[10];
double fee;

public:
student()
{
cout << "Enter the RollNo:";
cin >> rno;
cout << "Enter the Name:";
cin >> name;
cout << "Enter the Fee:";
cin >> fee;
}

void display()
{
cout << endl << rno << "\t" << name << "\t" << fee;
}
};
int main()
{
student s; // constructor gets called automatically when
        // we create the object of the class
s.display();
```

```
return 0;
}
```

**Output**

Enter the RollNo:Enter the Name:Enter the Fee:

6.95303e-310

```
// defining the constructor outside the class

#include <iostream>
using namespace std;
class student {
int rno;
char name[50];
double fee;
public:
student();
void display();
};
student::student()
{
cout << "Enter the RollNo:";
cin >> rno;

cout << "Enter the Name:";
cin >> name;

cout << "Enter the Fee:";
cin >> fee;
}
void student::display()
{
cout << endl << rno << "\t" << name << "\t" << fee;
}

int main()
{
student s;
s.display();

return 0;
}
```

**Output:**

Enter the RollNo: 30

Enter the Name: ram

Enter the Fee: 20000

30 ram 20000

# Constructors type

**Default Constructors**

A constructor without any arguments or with the default value for every argument is said to be the Default constructor.

A constructor that has zero parameter list or in other sense, a constructor that accept no arguments is called a zero argument constructor or default constructor.

If default constructor is not defined in the source code by the programmer, then the compiler defined the default constructor implicitly during compilation.

If the default constructor is defined explicitly in the program by the programmer, then the compiler will not defined the constructor implicitly, but it calls the constructor implicitly.

Example:

```
#include<iostream>
using namespace std;
class student
{
    int rno;
    char name[50];
    double fee;
    public:
    student()                                   // Explicit Default constructor
    {
      cout<<"Enter the RollNo:";
      cin>>rno;
      cout<<"Enter the Name:";
      cin>>name;
      cout<<"Enter the Fee:";
      cin>>fee;
```

```
    }

    void display()
    {
      cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;
    }
};

int main()
{
    student s;
    s.display();
    return 0;
}
```

## Parameterized Constructor

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

Let's see the simple example of C++ Parameterized Constructor.

```
1.  #include <iostream>
2.  using namespace std;
3.  class Employee {
4.    public:
5.      int id;//data member (also instance variable)
6.      string name;//data member(also instance variable)
7.      float salary;
8.      Employee(int i, string n, float s)
9.      {
10.       id = i;
11.       name = n;
12.       salary = s;
13.     }
14.     void display()
```

```
15.         {
16.             cout<<id<<" "<<name<<" "<<salary<<endl;
17.         }
18.     };
19.     int main(void) {
20.         Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
21.         Employee e2=Employee(102, "Nakul", 59000);
22.         e1.display();
23.         e2.display();
24.         return 0;
25.     }
```

**Output:**

```
101  Sonoo  890000
102  Nakul  59000
```

## *What distinguishes constructors from a typical member function?*

1. Constructor's name is the same as the class's
2. Default There isn't an input argument for constructors. However, input arguments are available for copy and parameterized constructors.
3. There is no return type for constructors.
4. An object's constructor is invoked automatically upon creation.
5. It must be shown in the classroom's open area.
6. The C++ compiler creates a default constructor for the object if a constructor is not specified (expects any parameters and has an empty body).

By using a practical example, let's learn about the various constructor types in C++. Imagine you visited a store to purchase a marker. What are your alternatives if you want to buy a marker? For the first one, you ask a store to give you a marker, given that you didn't specify the brand name or colour of the marker you wanted, simply asking for one amount to a request. So, when we just said, "I just need a marker," he would hand us whatever the most popular marker was in the market or his store. The default constructor is exactly what it sounds like! The second approach is to go into a store and specify that you want a red marker of the XYZ brand. He will give you that marker since you have brought up the subject. The parameters have been set in this instance thus. And a parameterized constructor is exactly what it sounds like! The third one requires you to visit a store and declare that you want a

marker that looks like this (a physical marker on your hand). The shopkeeper will thus notice that marker. He will provide you with a new marker when you say all right. Therefore, make a copy of that marker. And that is what a copy constructor does!

## What are the characteristics of a constructor?

1. The constructor has the same name as the class it belongs to.
2. Although it is possible, constructors are typically declared in the class's public section. However, this is not a must.
3. Because constructors don't return values, they lack a return type.
4. When we create a class object, the constructor is immediately invoked.
5. Overloaded constructors are possible.
6. Declaring a constructor virtual is not permitted.
7. One cannot inherit a constructor.
8. Constructor addresses cannot be referenced to.
9. When allocating memory, the constructor makes implicit calls to the new and delete operators.

## What is a copy constructor?

A member function known as a copy constructor initializes an item using another object from the same class-an in-depth discussion on Copy Constructors.

Every time we specify one or more non-default constructors (with parameters) for a class, we also need to include a default constructor (without parameters), as the compiler won't supply one in this circumstance. The best practice is to always declare a default constructor, even though it is not required.

A reference to an object belonging to the same class is required by the copy constructor.

```
Sample(Sample &t)
{
id=t.id;
}
```

### Destructor:

A destructor is **a member function that is invoked automatically when the object goes out of**

**scope or is explicitly destroyed by a call to delete** . A destructor has the same name as the class, preceded by a tilde ( ~ ). For example, the destructor for class String is declared: ~String()

**What is a destructor?**

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

- Destructor is also a special member function like constructor. Destructor destroys the class objects created by constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object create by constructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when object goes out of scope.
- Destructor release memory space occupied by the objects created by constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

**Syntax:**

Syntax for defining the destructor within the class

~ <class-name>()

{

}

Syntax for defining the destructor outside the class

<class-name>: : ~ <class-name>()

{

}

```
// Example:

#include<iostream>
using namespace std;

class Test
{
  public:
```

```
    Test()
    {
        cout<<"\n Constructor executed";
    }

    ~Test()
      {
        cout<<"\n Destructor executed";
      }
};
main()
{
    Test t;

   return 0;
}
```

**Output**

Constructor executed

Destructor executed