Constructors and Destructors

- Multiple Constructors in Class
- Constructors with Default Arguments
- Destructors

Multiple Constructors in Class

- In C++, we can use multiple constructors in one class,is called constructor overloading.
- All constructors used in class have same name as of class, only difference is in number of arguments passed to that constructors.
- For example

```
sample();      // Default Constructor
sample(int x, int y);// Parameterized constructor
sample(sample & s);  // Copy Constructor
```

Multiple Constructors in Class

```
class sample
{
        int a;
        int b;
    public:
        sample()
        {  a=0;     b=0;       }
        sample(int x,int y)
        {  a=x;  b=y;    }
        sample(sample &s)
        {   a=s.a;   b=s.b;
        }
        void display()
        {
            cout<<a<<"  "<<b<<endl;
        }
};
```

Multiple Constructors in Class

```
class sample
{
        int a;
        int b;
    public:
        sample()
        {  a=0;     b=0;       }
        sample(int x,int y)
        {  a=x;  b=y;     }
        sample(sample &s)
        {   a=s.a;   b=s.b;
        }
        void display()
        {
            cout<<a<<"  "<<b<<endl;
        }
};
```

```
int main()
{
    sample s1;
    sample s2(10,20);
    sample s3(s2);
    s1.display();
    s2.display();
    s3.display();

}
```

Output:

```
0 0
10 20
10 20
```

Constructors with Default Arguments

In C++, it is possible to define constructors with default arguments, for example, the constructor sample can be defined as

```
sample(int x, int y=0);
```

The default value of argument y is 0, then the calling statement would be

```
Sample S(10);
```

In this case, x=10 and y=0 will be assigned and passed to constructor function.

However, the calling statement

```
Sample s(10,20);
```

Will assign x=10 and y=20; In this, the zero value will be overwritten with value 20.

Constructors with Default Arguments

```
class sample
{
    private:
        int a;
        int b;
    public:
        sample(int x,int y=0)
        {
            a=x;
            b=y;
        }
        void display()
        {
            cout<<a<<"  "<<b<<endl;
        }
};
```

```
int main()
{
    sample s1(10);
    sample s2(10,20);
    s1.display();
    s2.display();
}
```

Output:

```
10 0
10 20
```

Destructors

A *destructor*, as the name implies, is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde. For example, the destructor for the class integer can be defined as shown below:

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

A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program (or block or function as the case may be) to clean up storage that is no longer accessible. It is a good practice to declare destructors in a program since it releases memory space for future use.

Destructors

```
class Employee
 {
   public:
        Employee()
        {
            cout<<"Constructor Invoked"<<endl;
        }
        ~Employee()
        {
            cout<<"Destructor Invoked"<<endl;
        }
};
```

```
int main()
{
    //creating an object of Employee
    Employee e1;
    //creating an object of Employee
    Employee e2;
    return 0;
}
```

Destructors

```
class sample
{       int a;
        int b;
    public:
        sample(int x,int y)
        {
            a = x; b = y;
        }
        void display()
        {
            cout<<a<<"  "<<b<<endl;
        }
        ~sample()
        {
            cout<<"Destructor Invoked";
        }
};
```

```
int main()
{
    sample s(10,20);
    s.display();
}
```