Chapter 4 : Constructors & Destructor

Constructors

<u>What is constructor?</u>

A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class because it does not have any return type.

<u>A constructor is different from normal functions in following ways</u> :

- Constructor has same name as the class itself
- Constructors don't have return type
- A constructor is automatically called when an object is created.
- It must be placed in public section of class.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).

1. **<u>Default Constructors</u> :** Default constructor is the constructor which doesn’t take any argument. It has no parameters.

2. <u>Parameterized Constructors</u> : It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object.

Uses of Parameterized constructor :
a. It is used to initialize the various data elements of different objects with different values when they are created.
b. It is used to overload constructors.

3. **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.

Whenever we define one or more non-default constructors (with parameters) for a class, a default constructor (without parameters) should also be explicitly defined as the compiler will not provide a default constructor in this case.

However, it is not necessary but it’s considered to be the best practice to always define a default constructor.

Copy Constructor

What is a copy constructor?

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

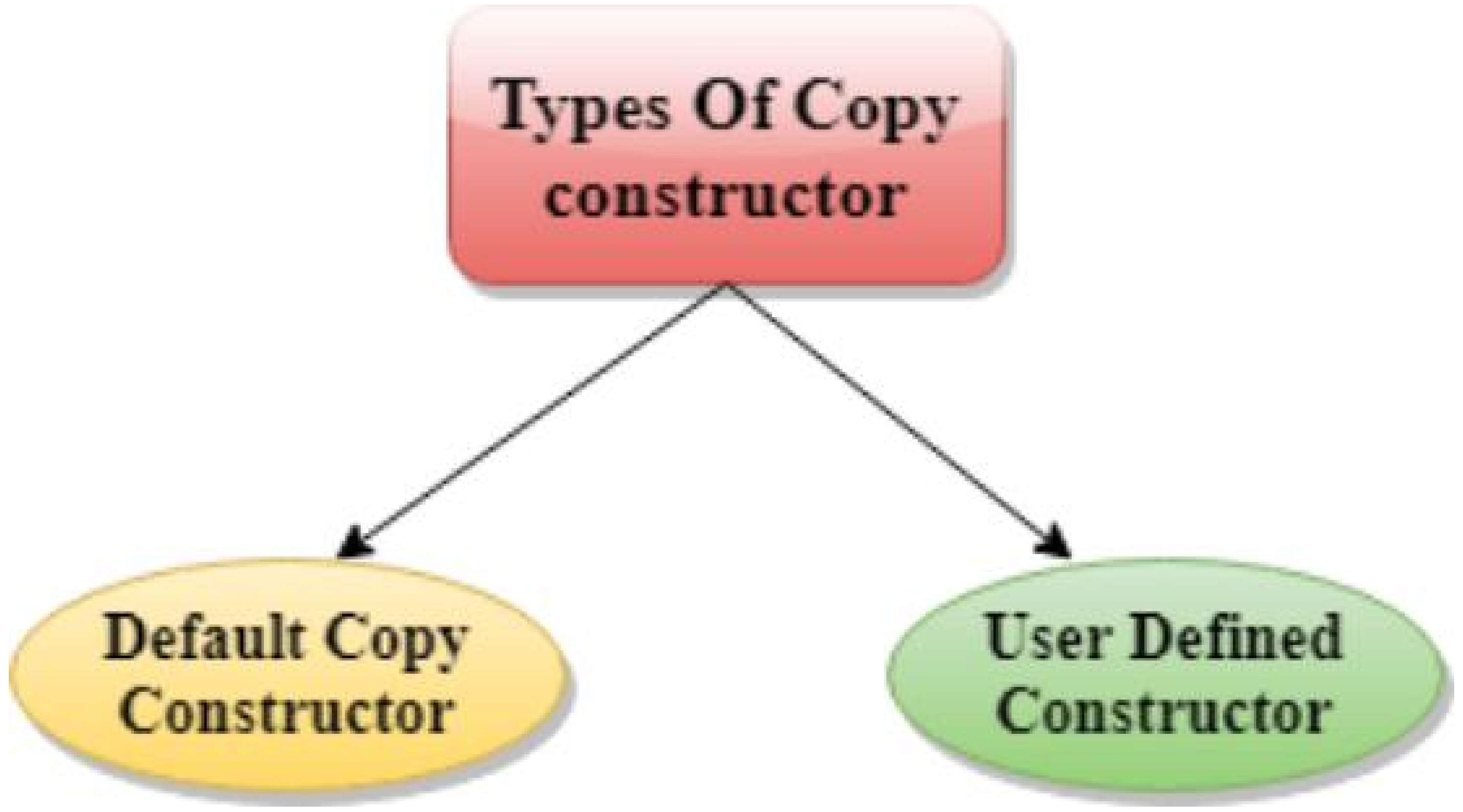
A copy constructor has the following general function prototype :

```
ClassName (const ClassName &old_obj);
```

Copy Constructor is of two types

Default Copy constructor : The compiler defines the default copy constructor. If the user defines no copy constructor, compiler supplies its constructor.

User Defined constructor : The programmer defines the user-defined constructor.


Types Of Copy constructor
Default Copy Constructor
User Defined Constructor

Following example of copy constructor :

```
#include<iostream>
using namespace std;
class Point
{
 private:
    int x, y;
 public:
    Point(int x1, int y1) { x = x1; y = y1;
   }
 // Copy constructor
    Point(const Point &p1) {x = p1.x; y = p1.y; }
    int getX()            {  return x; }
    int getY()            {  return y; }
};
int main()
{
    Point p1(10, 15); // Normal constructor is called here
Point p2 = p1; // Copy constructor is called here
    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
    return 0;
}
```

Output :

```
p1.x = 10, p1.y = 15
p2.x = 10, p2.y = 15
```

When is copy constructor called ?

1. When an object of the class is returned by value.
2. When an object of the class is passed (to a function) by value as an argument.
3. When an object is constructed based on another object of the same class.
4. When the compiler generates a temporary object.

Dynamic Constructors

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.

For Example :

```
#include <iostream>
using namespace std;
class icomputercoding {
    int* p;
```

```
public:
    // default constructor
    icomputercoding()
    {
        // allocating memory at run time
        // and initializing
        p = new int[3]{ 1, 2, 3 };
        for (int i = 0; i < 3; i++) {
      cout << p[i] << " ";
```

```
 }
     cout << endl;
    }
};
int main()
{
    // five objects will be created
    // for each object
    // default constructor would be called
    // and memory will be allocated
 // to array dynamically
    icomputercoding* ptr = new icomputercoding[5];
}
```

Output :

```
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
```

Destructors

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.
- The thing is to be noted here, if the object is created by using new or the constructor uses new to allocate memory which resides in the heap memory or the free store, the destructor should use delete to free the memory.

Syntax :

~constructor-name();

Properties of Destructor :

- Destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.
- It has no return type not even void.
- An object of a class with a Destructor cannot become a member of the union.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of destructor.

For Example :

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Constructor Invoked"<<endl;
 }
        ~Employee()
        {
       cout<<"Destructor Invoked"<<endl;
        }
};
int main(void)
{
   Employee e1; //creating an object of Employee
 Employee e2; //creating an object of Employee
   return 0;
}
```

Output :

```
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
```