Subject Name: **Object Oriented Programming Using C++**

Subject Code: **BCA-301 N**

Subject Topic: **Member Function Definition Outside and Introduction to Constructor**

Abhishek Dwivedi

Assistant Professor

Department of Computer Application

UIET, CSJM University, Kanpur

Functions definition : Outside the class

- To define a function outside of a class, scope resolution operator **::** is used.
- Syntax for declaring function outside of class

```
class class_name
{   ........
    ........
public:
  return_type  function_name (args); //function declaration
};
//function definition outside class
return_type class_name :: function_name (args)
{
 ...........; // function definition
}
```

Example

```
#include <iostream>
class smallobj
{
private:
      int somedata;
public:
   void setdata(int d);
   void showdata();
 };
Void  smallobj :: setdata(int d)
{
   somedata=d;
}
Void smallobj :: showdata()
{
 cout << "Data is : " << somedata << endl;
 }
```

Passing an object within the class member function as an argument

```
#include <iostream>
class A
{
public:
  int n=100;
  char ch='A';
   void disp(A a)
  {
     cout<<a.n<<endl;
     cout<<a.ch<<endl;
  }
};

int main()
 {
     A obj;
    obj.disp(obj);
     return 0;
 }
```

Constructor

- A constructor is a 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.

A constructor is different from normal functions in following ways:

a. Constructor has same name as the class itself

b. Constructors don't have return type

c. A constructor is automatically called when an object is created.

d. If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).

Example:

```
class constructorDemo
{
    public:
            int num;
            char ch;
            constructorDemo()
               { num = 100;
                 ch = 'A';
               }
};
int main()
{
    ConstructorDemo  obj;
    cout<<"num: "<<obj.num<<endl;
     cout<<"ch: "<<obj.ch;
    return 0;
}
```

Types of Constructors:

There are three types of Constructors:

- **Default**
- **Parameterized**
- **Copy**

Default Constructor: A default constructor doesn't have any arguments

```
#include <iostream>
class Website
{
        public:
         Website()            //Default constructor
          {
             cout<<"Welcome to Beginners Book"<<endl;
           }
};
void main()
{
Website obj1;
Website obj2;
return 0;
}
```

Parameterized Constructor:

Constructors with parameters are known as Parameterized constructors. These type of constructor allows us to pass arguments while object creation.

```
#include <iostream>
class Add
{
    public:
        Add(int num1, int num2)       //Parameterized constructor
        {
            cout<<(num1+num2)<<endl;
         }
};
int main(void)
{
Add obj1(10, 20); // One way of creating object. This is known as implicit call to the constructor

Add obj2 = Add(50, 60); //Another way of creating object. This is known as explicit calling the constructor
return 0;
 }
```

Copy constructor

- Copy Constructor is a type of constructor which is used to create a copy of an already existing object of a class type. It is usually of the form **X (X &objectname)**, where X is the class name.
- Syntax of Copy Constructor:

```
Classname(const  classname &objectname)
 {
        . . . .
}
```

- As it is used to create an object, hence it is called a constructor. And, it creates a new object, which is exact copy of the existing copy, hence it is called **copy constructor**.

Example

```
Class copyconstructor
{
    private:
        int x, y; //data members
     public:
        copyconstructor(int x1, int y1)
                { x = x1; y = y1; }
        copyconstructor (const  copyconstructor  &sam)          /* Copy constructor */
                { x = sam.x; y = sam.y; }
        void display()
                { cout<<x<<" "<<y<<endl; }
 };
int main()
 {      copyconstructor       obj1(10, 15); // Normal constructor
        copyconstructor       obj2 = obj1; // Copy constructor
        cout<<"Normal constructor : ";
         obj1.display();
        cout<<"Copy constructor : ";
         obj2.display();
        return 0;
}
```

References:

- www.studytonight.com
- www.tutorialpoint.com
- www.geeksforgeeks.org
- “Object oriented programming in C++” Robert Lafore
- “Object oriented programming with C++”, E.Balagurusamy