Constructors & Destructor

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

- C++ provides a special member function, whose task is to Initialize the objects automatically, when it is created. This function known as Constructor.
- Constructor has same name as that of class and it does not have any return type.
- It is called constructor because it construct the values of data members of the class.

Declaration & definition of Constructor:

```
Class A
{
int m , n;
Public:
A(void);                    //    Constructor declared
…….
…….
};
A::A(void)                         //    Constructor defined
{
M=0; n=0;
}
```

Characteristics of Constructor:

1) Constructor is used for Initializing the values to the data members of the Class.
2) Constructor is that whose name is same as name of class.
3) Constructor gets Automatically called when an object of class is created.
4) Constructors never have a Return Type even void.
5) Constructor are of Default , Parameterized and Copy Constructors.

Types of Constructor:

Constructors are of three types :

1) Default Constructor
2) Parameterized Constructor
3) Copy Constructor

Default Constructor

The constructor which doesn't take any argument.

Syntax: constructor_name ();

Declaration & Definition of Default Constructor:

```
Class A
{
int m , n;
Public:
A(void);                          //    Constructor declared
…….
…….
};
A::A(void)                                    //    Constructor defined
{
M=0; n=0;
}
```

Example:

```
class Cube
{
int side;
public:
Cube()
 {
 side=10;
}};
int main()
{
Cube c;
 cout << c . side;
}
```

In this case, as soon as the object is created the constructor is called which initializes its data members.

A default constructor is so important for initialization of object members , that's why even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

Parameterized Constructor

The constructor that can take parameters are called Parameterized Constructor.

Using this Constructor user can provide different values to data members of different objects, by passing the appropriate values as argument.

Declaration & Definition of Parameterized Constructor:

```
Class A
{
int m , n;
Public:
A(int x, int y);                         //    Constructor declared
…….
…….
};
A::A(void)                               //    Constructor defined
{
m=x; n=y;
}
```

Example:

```
        class Cube
       {
        int side;
        public:
       Cube(int x)
       {
        side=x
         }};
       int main()
       {
        Cube c1(10);
        Cube c2(20);
        cout << c1.side;
         cout<<c2.side;
}
```

In this case, by using parameterized constructors we initialized 2 objects with user defined values.

We can have any number of parameters in a constructor.

When we used parameterized constructor, we must pass the initial value as arguments to the constructor, when the object declared. This can be done in 2 ways:

1) By calling constructor implicitly.

Ex: A a (5,10);

2) By calling constructor explicitly.

Ex: A a = A(5,10);

Example:

```
      class A
   {
      int a, b;
      public:
           A(int i, int j)
       {
           a=i;
           b=j;
       }
           void show()
       {
           cout<< a << " " << b;
        }
   };
          void main()
   {
          A ob(3, 5);
         A ob1 = A(5,4);
          ob . show();
          ob1 . show();
   }
```

Output:

```
3,5
5,4
```

Copy Constructor

When a constructor accept a reference to its own class as a parameter, is called copy constructor .

Copy Constructor is used for Copying the values of class object into an another object of class. To Copy the values we have to pass the name of object whose values we wants to copying and when we are using or passing an Object to a Constructor then we must have to use the & Ampersand or Address Operator.

Declaration & Definition of Copy Constructor:

```
Class A
{
int  a;
public:
A( A &);                                    //    Constructor declared
};
A::A(A &x)                                  //    Constructor defined
{
a=x . a;
}
```

Example:

```
class A
{
int id;
public:
A(int a)
{
id=a;
}
A(A &x)
{
id=x.id;
}
void display()
{
cout<< id;
}
};
int main()
{
A obj (100);
B obj1(obj);
cout<< obj . display()<<endl;
cout<< obj1 .display()<<endl;
}
```

Output:

```
100
100
```

Constructor Overloading in C++

Like Member function, Constructors can also be overloaded. When two or more constructors function defined in a class, is called Constructor Overloaded .

Example:

```
Class A
{
int  m , n;
public:
A()                                          //   Default Constructor
{            m=0;      n=0;        }
A(int a, int b)                              //   Parameterized Constructor
{           m=a;       n=b;        }
A( A & x);                          //    Copy  Constructor
{           m=x . m;     n=x . N;}
};
```

Program :

```
class A
{
    int a , b ;
    public:
    A()
    {
    a=100;      //   Default Constructor
    b=200;
    }
    A(int i,int j)
    {
    a=i;       //   Parameterized Constructor
    b=j;
    }
    A(A &x , A &y)
    {
        a = x.a;   //   Copy Constructor
        b = x.b;
    }
```

```
 void Display()
    {
    cout<<"Values :"<<a<<"\t"<<b<<endl;
    }
};

int main()
      {
       A obj;
       A obj1(10,20);
       A obj2(obj1);

       obj.Display();
       obj1.Display();
       obj2.Display();
      }
```

Output:

```
100 200
10  20
10  20
```

Constructor with default arguments in C++

It is possible to define constructors with default arguments.

Example:

```
Class A
{
int  m , n;
public:
A(int a, int b=8)                                    //   Parameterized Constructor
{
           m=a;      n=b;
}
```

Program:

```
class student
{
int rollno;
float marks;
public:
student(int x=10,float y=40)
{
Rollno =x;
marks=y;
}
void display()
{
cout<<"Roll no: "<<rollno<<endl;
cout<<"Marks: "<<marks<<endl;
}
};
```

```
void main()
{
student s1;
student s2(1,60);
clrscr();
cout<<"Output using default constructor
      arguments:\n";
s1.display();
cout<<"Output without default constructor
      arguments:\n";
s2.display();
}
```

Output:

```
10  40
1   60
```

Dynamic Initialization of Objects in C++

Class object can be initialized dynamically too. It means that the initial value of an object may be provided during run time.

The main advantage of dynamic initialization is that user can provide various initialization formats, using overloaded constructors. This provides the flexibility of using different format of data at run time depending upon the situation.

Program:

```
Class add
{
      int a , b , c ;
      Public:
      add( ) { }
      add (int  a1,int b1)
      {
            a=a1, b=b1;
            c= a + b ;
      }
      void display( )
      {
            cout<<“sum is”<<c;
      }
};
```

```
void main()
{
      int x , y ;

      cout<<"Enter the value \n";
      cin>>x>>y;
      add d(x , y);
      d. display();
}
```

Output:

```
Enter the value
3
2
Sum is 5
```

Dynamic Constructor in C++

The constructors can also be used to allocate memory while creating objects i.e. at the run time. This will enable the system to allocate the right amount of memory for each object when the objects are not the same size, hence save the memory.

The memory allocation to objects is allocated with the help of 'new' operator.

By using this constructor, we can dynamically initialize the objects.

```
class A
{
 int * p;
 public:
 A()
 {
  p=new int;
  *p=10;
 }
 A(int v)
 {
  p=new int;
  *p=v;
 }
 int display()
 {
return(*p);
 }
};
```

```
void main()
{


A obj;
A obj1(9);

cout<<"The value of object obj‘ s p is:";
cout<<obj.dis();
cout<<"\n The value of object 0bj1‘ s p
      is:"<<obj1.dis();


}
```

Output:

```
The value of object obj‘ s p is
10
The value of object obj1‘ s p is
9
```

Destructor in C++

Destructor is used to destroy the objects that have been created by a constructor. Like a constructor its name same as the class name but is preceded by a tilde.

It is called automatically by the compiler when the object goes out of scope.

Syntax :

```
~A()
{
}
```

Characteristics Of Destructor:

1) A destructor never takes any argument nor does it return any value.
2) It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible.

Whenever new is used to allocate memory in the constructors, we should use delete to free that memory.

Program: 1

```
class A
{
public:
int x , y ;

A(int i, int j)   /// parameterized constructor
{
x = i;
y = j;
}

~A(); // destructor
};

A::~A()
{
cout << "Destructing object whose x & y val
   is " << x <<"\t"<< y <<" \n";
}}
```

```
int main()
  {
  A t1(5,10);
  A t2 = A(19,20);
  cout<<"for implicitly call \n";
  cout << t1.x << " " << t1.y << "\n";

  cout<<"for explicitly call \n";
  cout << t2.x << " " << t2.y << "\n";
   }
```

Output:

```
for implicitly call
5   10
for explicitly call
19 20
Destructing object whose x & y value is
19 20
Destructing object whose x & y value is
5   10
```

ww

Program:2

```
class integer
      {
            private:
            int a;
            public:
                              integer()
                     {
                               a=0;
cout<<endl<<"Default Constructor Called. : ";
                     }
                              integer(int i)
                     {
                               a=i;
cout<<endl<<"Parameterized Constructor  called. : ";
                     }
                             ~integer()
                     {
                          cout<<endl<<"Destructor Called.";
                      }
```

```
  void get()
{
  cout<<"Enter the value of
       a:" <<endl;
  cin>>a;
}

void show()
{
cout<<" Value of a is :"<<"\t"<<a;
}
};

  void main()
          {
                 integer x, y(10);
                   x . get();
                   x . show();
                   y . show();

          }
```