Lecture 7

CONSTRUCTORS AND DESTRUCTORS

➢ Introduction:

• It is sometimes convenient if an object can initialize itself when it is first created, without the need
to make a separate call to member functions.
• Automatic initialization is carried out using special member functions called constructors.

➢ Constructors:

• ***A Constructor is a special member function that is called automatically when an object is created.***
• The purpose of a constructor is to mainly initialize the member variables of a class.
• The general syntax of a the constructor in C++ is:
In the above example class declaration, class **Sum** has a member function **Sum( )** with the same name of the class and which provides initial value to its data member s.

➢ Characteristics of Constructor:

• The name of the constructor is the same as the name of the class.
Example: In the above class, name of the class is **Sum** and the function name is **Sum**.
• A Constructor, even though it is a function, has no return type, i.e. it is neither a value-returning
function nor a void function.
Example: **Sum ( )** function has no return type and not even void.
• The constructor should be declared in the ***public*** section.
• Constructors are executed automatically i.e. they are never invoked. They are executed when a class object is created.

• A class can have more than one constructor. However all constructor of a class should have the same name.
• It is not possible to refer to the address of the constructors.
• The constructors make implicit calls to the operator new and delete when memory allocation is required.

• **Example: Program to demonstrate how constructor is automatically executed at the time of object creation.**

```
#include<iostream.h>
#include<conio.h>
class Student
{
public:
Student( )
```

```
{
cout<<"Constructor called automatically";
cout<<"at the time of object creation"<<endl;
}
};
void main( )
{
Student S1;
Student S2;
Student S3;
}
```

OUTPUT:

```
Constructor called automatically at the time of object creation
Constructor called automatically at the time of object creation
Constructor called automatically at the time of object creation
```

- **Example: Program to demonstrate how a constructor is use to initialize data member of an object.**

```
#include<iostream.h>
#include<conio.h>
class Number
{
private:
int a;
public:
Number ( )
{
cout<<"I am in the Constructor";
a = 100;
}
void display( )
{
cout<<"Value of a is ="<<a;
}


void main( )
{
Number N;
N.display();
}
```

OUTPUT:

```
I am in the Constructor
Value of a is = 100
```

➢ Need for a Constructor:

- Constructors are named as constructors because they are getting called when an object is constructed.
- The use of a constructor can be cleverly done especially in those problems where it is necessary to initialize certain data members compulsorily.
- Instead of having separate member functions for initializing we can perform those operations inside the constructor itself.

- **Example: A Program to find the sum of N natural numbers using a class constructor.**

```
#include<iostream.h>
#include<conio.h>
class Sum
{
private:
int n, s;
public:
Sum ( )
{
s = 0;
}
void readdata( )
{
cout<<"Enter the input limit"<<endl;
cin>>n;
}
void display( )
{
for(int i =1; i<=n; i++)
s = s + i;
cout<<"Sum of Natural numbers ="<<s;
}
};
void main( )
{
Sum S1;
S1.readdata( );
S1.display( );
}
```

OUTPUT:

```
Enter the input limit
10
Sum of Natural numbers = 55
```

➢ Types of constructor:

- Constructors are normally classified as follows:
  - Default Constructors.
  - Parameterized Constructors
  - Copy Constructors.

➢ Default Constructors:

- ***A default constructor is a special member function which is invoked by the C++ compiler without any argument for initializing the object of a class.***

- It is also called as zero argument constructors.
- Some of the features of the default constructors are:
  - A default constructor function initializes the data member with no argument.
  - It can be explicitly written in the public section of the class.
  - In case, default constructor is not defined in a program, the C++ compiler automatically generates it in a program.
  - The purpose of the default constructor is to construct a default object of the class type.

- The general format of default constructor is as follows:

Syntax	**Example**

class Class_Name
{
public:
Class_Name( )
{
…….
}
};
	class Number
	{
	public:
	Number( )
	{
	n = 0;
	}
	};

- **Example: A program to display N natural numbers using a class default constructor.**

```
#include<iostream.h>
#include<conio.h>
class Number
{
private:
int n;
public:
  Number ( )                    //Default Constructor with no arguments
  {
    n = 0;
  }
void readdata( )
{

cout<<"Enter the input limit"<<endl;
```

```
cin>>n;
}
void display( )
{
for( i =1; i<=n; i++)
cout<<"Natural numbers ="<< i<<"\t";
}
};
void main( )
{
Sum S1;
S1.readdata( );
S1.display( );
}
```

OUTPUT:

```
Enter the input limit
10
Natural numbers = 1 2 3 4 5 6 7 8 9 10
```

- **Some disadvantages of default constructors are:**
  - When many objects of the same class are created, all objects are initialized to same set of values by default constructors.
  - It is not possible to initialize different objects with different initial values using default constructors.

➢ Parameterized Constructors:

- ***A constructor that takes one or more arguments is called parameterized constructor.***
- Using this constructor, it is possible to initialize different objects with different values.
- Parameterized constructors are also invoked automatically, whenever objects with arguments are created. The parameters are used to initialize the objects.
- The keyword ***inline*** is used to define inline function.
- The general format of parameterized constructor is as follows:

Syntax	**Example**

class Class_Name
{
public:
Class_Name(argu1, argu2….)
{
…….
}
};
	class MAX
	{
	public:
	MAX(int a, int b)
	{
	if (a > b)
	big = a;
	else
	big = b;
	}
	};

• **Some of the features of the parameterized constructors are:**
- The parameterized constructors can be overloaded.
- For an object created with one argument, constructor with only one argument is invoked and executed.
- The parameterized constructor can have default arguments and default values.

✓ **Invoking Constructors:**

• A Constructor is automatically invoked by C++ compiler with an object declaration. The constructor can be invoked through the following methods.
- Implicit Call
- Explicit Call
- Initialization at the time of declaration with " = " operator.

✓ **Implicit Call:**

• An Implicit call means the declaration of the object is followed by argument list enclosed in parenthesis.

• **Example: Program to initialize the data members using implicit declaration**

```
#include<iostream.h>
#include<conio.h>
class num
{
private:
int a, b;
public:
  num ( int m, int n)
  {                              //Parameterized Constructor
a = m;
b = n;
}
void display( )
{
cout<<" a = " << a <<" b = " << b;
}
};
void main( )
{
  num obj1(10, 20);
  num obj2(40, 50);          //Implicit Call
  obj1.display( );           //Implicit Call
  obj2.display( );
}
```

OUTPUT:

a = 10 b = 20
a = 40 b = 50

✓ **Explicit Call:**

• In explicit call, declaration of an object is followed by assignment operator, constructor name and
argument list enclosed in parenthesis.

• **Example: Program to initialize the data members using explicit declaration**

```
#include<iostream.h>
#include<conio.h>
class num
{
private:
int a, b;
public:
 num ( int m, int n)                    //Parameterized Constructor
 {
  a = m;
  b = n;
}
void display( )
{
cout<<" a = " << a <<" b = " << b;
}
};
void main( )
{
 num obj1 = num(10, 20);
 num obj2 = num(40, 50);          //Explicit Call
 obj1.display( );                 //Explicit Call
 obj2.display( );
}
```

OUTPUT:

a = 10 b = 20
a = 40 b = 50

✓ **Initialization of object during declaration with assignment operator " = ":**

• This method is used for the constructor with exactly one argument. In this method declaration is followed by assignment operator and value to be initialized.

• **Example: Program to initialize objects using assignment operator.**

```
#include<iostream.h>
#include<conio.h>
class num
{
```

```
private:
int a;
public:
  num ( int m)                    //Parameterized Constructor
  {
a = m;
}
void display( )
{
cout<< a << endl ;
}
};
void main( )
{
num obj1 = 100
num obj2 = 200
cout<<"Object1 = " ;
obj1.display( );
cout<<"Object2 = " ;
obj2.display( );
}
```

OUTPUT:
Object1 = 100
Object2 = 200

➢ Copy Constructors:

• ***Copy constructor is a parameterized constructor using one object can be copied to another object.***

• Copy Constructors are used in the following situations:
  o To initialize an object with the values of already existing objects.
  o When objects must be returned as function values.
  o To state objects as by value parameters of a function.

• Copy Constructor can accept a single argument of reference to same class type. The argument must be passed as a constant reference type.

• The general format of copy constructor is as follows:

Syntax	**Example**

class Class_Name
{
public:
Class_Name(Class_Name &ptr)
{
…….
}
};
	class Number
	{
	public:
	Number(int n)
	{ a = n; }
	Number(Number & X)
	{
	a = X.a;

	cout<<"Copy Constructor invoked";
	}
	};

- **Note that:**
  - Copy constructor is not invoked explicitly.
  - Copy constructors are invoked automatically when a new object is created and equated to an already existing object in the declaration statement itself.

Example:	x x	a1; a2 = a1;	//Default Constructor //Copy Constructor
o When a new object is declared and existing object is passed as a parameter to it in the declaration, then also copy constructor is invoked.			
Example:	x x	a1(100, 200); a2(a1);	//Parameterized Constructor //Copy Constructor invoked

  - When object is passed to a function using pass by value, copy constructor is automatically called.
  - Copy constructor is invoked when an object returns a value.

- **Example: Program to find factorial of a number using copy constructor.**

```
#include<iostream.h>
#include<conio.h>
class copy
{
private:
int var;
public:
copy ( int temp)
{
var = temp;
}
int calculate( )
{
int fact, i;
fact = 1;
for( I =1; i<=var; i++)
fact = fact * I;
return fact;
}
```

```
};
void main( )
{
int n;
cout<<"Enter the Number:";
cin>>n;
copy obj(n);
copy cpy = obj;
cout<<"Before Copying : "<< n<<"! =" <<obj.calculate( )<<endl;
cout<<"After Copying : "<< n<<"! =" <<cpy.calculate( )<<endl;
}
```

OUTPUT:

```
Enter the Number: 5
Before Copying: 5! = 120
After Copying: 5! = 120
**********
```