CONSTRUCTOR AND DESTRUCTOR

-J. V. Patil

Introductions to Constructors

- When a variable is declared and if it is not initialized, it contains the ***garbage*** value.
- The programmer needs to initialize variables with appropriate values.
- This can be done through the public member functions.
- In the first step the object is created and in the second step the data members are initialized through public member functions.
- It could be better if the initialization is done at the time the object is created or declared.
- This can avoid calling the public member function to initialize the data members. This initialization is possible with ***constructors***.

CONSTRUCTOR

- Constructor is a special function used to initialize class data members or we say constructor is used to initialize the objects of class.
- Whenever, an object is created, the constructor will be executed automatically and initialization of data member takes place.
- A Constructor is a special member as it has the same name as that of the class and is automatically invoked whenever an object of the class is created.

- Constructor Characteristics:
  - A constructor's name must be as the class name in which it is declared.
  - It does not have any return value.
  - They should be declared in public section.
  - It can have default argument.
  - It can not be inherited like other member functions.
  - It is automatically called when an object is created.
  - It can not be referred by its address.

- The syntax to define a constructor (inside class):

```
class  Class_name
{
    public:
            Class_name(parameter list)
        {
            //body of constructor
        }
};
```

Where, parameter list is optional

- The syntax to define a constructor (outside class):

```
class  Class_name
{
    public:
            Class_name(parameter list); //constructor prototype
};
Class_name :: Class_name(parameter list) //constructor definition
{
    //body of constructor
}
```

Where, parameter list is optional

EXAMPLE OF CONSTRUCTOR

```
#include<iostream>
using namespace std;
class A
{
    int number;
public:
     A(); //Constructor declared
};
A:: A()                   // Constructor definition
{
    number=10;
}
int main()
{
    A a;
    cout<<"Value of number variable is :"<<a.number;
}
```

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

TYPES OF CONSTRUCTOR

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

DEFAULT CONSTRUCTOR

- The constructor which does not accept any argument is called default constructor. In default constructor the argument list is void.
- Default constructor is also called as empty constructor because of it has no arguments.
- A default constructor is used to initialize all the objects of a class with the same values.
- There can be only one default constructor in a class.
- The default constructor is defined as, a constructor which does not contain any parameters/arguments.
- A default constructor can be called directly when an object of the class is created. If no constructor are created, compiler will create a default constructor by itself.

- Syntax of default constructor:

```
class class_name
{
    . . .
    public:
        class_name()
        {
            //body of constructor
        }
};
```

- For example:

```
student:: student()
{
    rollno=1;
    marks=10;
}
```

PARAMETERIZED CONSTRUCTOR

- Sometimes, it is essential to initialize the various data elements of different objects with different values when they are created. This is achieved by passing arguments to the constructor function when the objects are created.
- A constructor that receives arguments/parameters is called parameterized constructor.
- To create parameterized constructor, simply add parameters to the constructor function as the same way you do with normal function.

- Syntax of Parameterized Constructor

```
    class class_name
{
    …
    public:
    class_name(arg1,arg2,… argn)
    {
        //body of constructor
    }
};
```

- For example

```
student::student(int a, int b)
{
roll_no=a;
marks=b;
}
```

COPY CONSTRUCTOR

- These are special type of constructors which takes an object as argument and is used to copy values of data member of one object into another object.
- In C++, a new object of a class can also be initialized with an existing object of the same class. For this, the compiler of C++ calls the copy constructor. Copy constructor creates the copy of the passed object.
- Initialization of an object through another object is called copy constructor.
- A copy constructor is used to declare and initialize an object from another object.
- For example, the statement

  Student S(S1);

  would define the object S and at the same time initialize it to values of S1.
- A copy constructor takes a reference to object of the same class as itself as an argument.

- Syntax of Copy constructor:

```
class class_name
{
    …
    public:
    class_name(class_name &object name)
    {
        //body of the constructor
    }
};
```

- For Example:

```
student :: student(student &t)
{
roll_no=t.roll_no;
}
```

MULTIPLE OR OVERLOADED CONSTRUCTORS

- C++ allows defining multiple constructors with different number, data type or order of parameters in a single class using constructor overloading.
- The number of constructors can be defined for the same class with varying argument list is referred as overloaded constructor or multiple constructor.
- Syntax: In C++, a class can simultaneously have a default constructor, a parametrized constructor etc.

```
class class_name
{   …
public:
    class_name()
    {   //body of default constructor
    }
    class_name()
    {   //body of parameterized constructor
    }
};
```

- For example

```
class integer
{
   int m, n;
  public:
   integer() { m=0; n=0 }   //constructor 1
   integer(int a, int b)   //constructor 2
   { m=a; n=b; }
   integer(integer &i)    //constructor 3
   { m=i.m; n=i.n; }
};
```

Here, more than one constructor function is defined in the class, we can say constructor is overloaded.

CONSTRUCTOR WITH DEFAULT ARGUMENTS

- A function uses the same arguments number of times for some situations, for example while calculating the total price of the items, same discount rate may applies to all the particular products.
- Therefore, instead of supplying such arguments all the time, C++ allows us to define an argument whose value will be automatically used by compiler if it is not provided during the function call. This argument is called as default arguments.
- For example, constructor complex can be declared as follows:

```
complex(float real, float imag=0)
```

- Default value of argument is imag is zero. Then the statement

```
complex c(5.0);
```

assign the value 5.0 to the real variable and 0.0 to the imag variable(by default). However the statement

```
complex(2.0,3.0);
```

assigns the value 2.0 to the **real** and 3.0 to the **imag.** The actual parameter when specified overrides the default value.

DESTRUCTOR

- A destructor is a special member function that is automatically called when an object is destroyed.
- Destructor is a member function whose name is the same as the class name, preceded by a tilde character (~).
- For example, the destructor for the student class can be declared as

```
~student();
```

- Destructors are invoked by the compiler implicitly when the termination of program takes place.

- Syntax of destructor:

```
class class_name
{
    …
      public:
    class_name() {  }
    ~class_name()
    {
        //body of destructor
    }
};
```

- A destructor releases the resources and memory at run time to clean up the unused storage area.

CHARACTERISTICS OF DESTRUCTORS

- A destructor name must be same as the class name in which it is declared, preceded by a tilde symbol(~).
- It does not have any return value.
- It is declared as a public member function.
- It takes no argument and therefore cannot be overloaded.
- It is automatically calls when object is destroyed.

DIFFERENCE BETWEEN CONSTRUCTOR AND DESTRUCTOR

	Constructor	Destructor
Purpose	Constructor is used to initialize the instance of a class.	Destructors destroys the objects when they no longer needed.
When called	Constructor is called automatically, when a new object of a class is created.	Destructors are called when instance of a class is deleted or released.
Overloading	Overloading of constructor is possible.	Overloading of destructor is not possible.
Name	Constructor has the same name as class name.	Destructor also has same name as class name but with tilde (~) symbol.
Argument	Constructor can have arguments.	Destructor can not have any arguments.

Declaration syntax	Class_name(arguments if any)
	{
	  //Body of constructor
	}
		~Class_name()
		{
		//Body of destructor
		}

In numbers	There can be multiple constructor in the class	There is always a single destructor in the class.