Unit-2 Object Oriented Programming with C++

BCA SEM-3

Unit-2	Constructor, Operator overloading and Type conversion	16	18
	- Constructor – Types of constructor, characteristics of constructor, constructor overloading. - Destructor - Basic of operator overloading - Types of operator overloading-Unary, Binary - Operator overloading using member function & friend function		

BCA SEM-3
Assistant Professor: Vinod.M.Makwana
KBPCCS

Que: 1 what is constructor? Explain types of constructor with example

A constructor is a special member function whose task is to initialize the objects of the class.

It's a special because its name is the same as the name of the class.

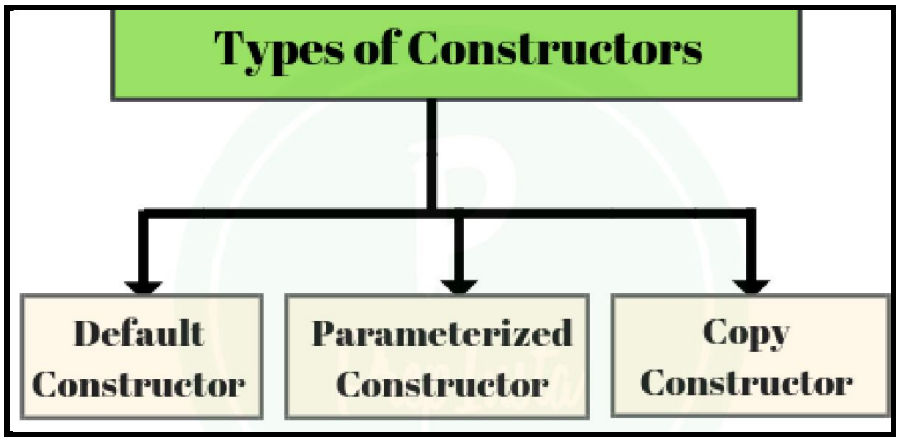
Constructor is invoked whenever an object of that class is created.

It is called constructor because it construct the values of data member of the class.

Important characteristics

1. The constructor function have some important characteristics are as below.
2. They should be declared in public section.
3. They are invoked automatically when object is created.
4. They do not have any return type even void hence they do not return any value.
5. They cannot be derived; they can be called through derived class.
6. They have default arguments as other programming languages.
7. They cannot be virtual.
8. We cannot refer to their address.

Types of constructor



[1] Default constructor

- Default constructor is the constructor which ***doesn't take any argument.***
- It has ***no parameter.***
- ***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 even ***if we do not define a constructor explicitly***, the compiler will ***provide a default constructor implicitly***

Syntax

```
class_name()
{
   // constructor Definition
}
```

Example:

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

Output

```
10
```

[2]Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Uses of Parameterized constructor:

- It is used to initialize the various data elements of different objects with different values when they are created.
- It is used to overload constructors.
- It can be called either implicit or explicit.
- Implicit means a constructor is called automatically, while a constructor is called by programmer is called explicitly called.

Syntax

```
class_name(parameter1, parameter2, ...)
{
   // constructor Definition
}
```

Example

```
class Square
{
   public:
   int side;
   Square(int a)
   {
      side=a;
   }
};
int main()
{
   Square s1(5);//implicit call
   Square s2(10);//implicit call
   cout << s1.side;
   cout << s2.side;

   square s3=square(15);//explicit call
   cout<<s3.side;
}
```

Output

```
5
10
15
```

[3]copy constructor

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

- Copy constructor is used to initialize the members of a newly created object by copying the members of an already existing object.
- Copy constructor takes a reference to an object of the same class as an argument.

Syntax:

ClassName (ClassName &obj);

Example

```
Sample(Sample &t)
    {
        id=t.id;
    }
```

- The process of initializing members of an object through a copy constructor is known as copy initialization.
- It is also called member-wise initialization because the copy constructor initializes one object with the existing object, both belonging to the same class on a member by member copy basis.
- The copy constructor can be defined explicitly by the programmer.
- It can be called either implicit or explicit.
- Implicit means a constructor is called automatically, while a constructor is called by programmer is called explicitly called.

When Copy Constructor is called

Copy Constructor is called in the following scenarios:

- When we initialize the object with another existing object of the same class type. For example, Student s1 = s2, where Student is the class.
- When the object of the same class type is passed by value as an argument.
- When the function returns the object of the same class type by value

```
#include <iostream>
using namespace std;
class A
{
  public:
                int x;
               A(int a)              // parameterized constructor.
               {
                        x=a;
                }
                A(A &i)              // copy constructor
               {
                        x = i.x;
               }
};

void main()
{
        A a1(20);             // Calling the parameterized constructor.
        A a2(a1);              //  Calling the copy constructor.
        A a3=a1;              //calling copy constructor
        A a4;
        a4=a3;                 //copy constructor not called.
        cout<<a2.x;

}
```

Que: 2 what is constructor overloading? Explain with example.

Defining more than one constructor with different arguments is called constructor overloading.

Overloaded constructors have the same name (name of the class) but the different number of arguments. Depending upon the number and type of arguments passed.

```
#include <iostream>
class Person
{
          int age;
         public:
         // 1. Constructor with no arguments
         Person()
        {
                 age = 20;
        }
         // 2. Constructor with an argument
         Person(int a)
         {
                  age = a;
          }
        //3.copy constructor
           Person(person &a)
         {
                  age = a.age;
          }
         int getAge()
         {
                  return age;
         }
};
void main() {
   Person p;//default constructor
   Person  p1(45);//parameterized constructor
   Person p3=p1;//copy constructor
   cout << "Person1  Age = " << person1.getAge() << endl;
   cout << "Person2  Age = " << person2.getAge() << endl;
   cout << "Person3  Age = " << person3.getAge() << endl;
 getch()
}
```

Que : 3 what is destructor? Give difference between constructor and destructor.

Destructor is a special 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.

Characteristics:

- Destructor is also a special member function like constructor because its name is same as the name of the class but is preceded by a tiled (~) symbol.
- Destructor destroys the class objects created by constructor.
- It is not possible to define more than one destructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when object goes out of scope.
- Destructor release memory space occupied by the objects created by constructor.
- In destructor, objects are destroyed in the reverse of an object creation.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of destructor.

Syntax:

```
~class_name()
 {
 }
```

```
#include<iostream.h>
#include<conio.h>
int count=0;
class demo
{
        public:
                demo()
                {
                        count++;
                        cout<<"Constructor is called for object:"<<count<<endl;
                }
                ~demo()
                {

                        cout<<"Destructor is called for object:"<<count<<endl;
                        count--;
                }
};
```

```
void main()
{
        clrscr();

        demo d1,d2,d3,d4,d5;
        {
                demo d6;
        }

}
```

Output:

```
Constructor is called for object:1
Constructor is called for object:2
Constructor is called for object:3
Constructor is called for object:4
Constructor is called for object:5
Constructor is called for object:6
Destructor is called for object:6
Destructor is called for object:5
Destructor is called for object:4
Destructor is called for object:3
Destructor is called for object:2
Destructor is called for object:1
```

Difference:

S. No.	Constructor	Destructor
1.	Constructor helps to initialize the object of a class.	destructor is used to destroy the objects of the class.

2.	It is declared as
	className(arguments if any)
	{
	// Constructor's Body
	}
		Whereas it is declared as
		~ className(no arguments)
		{
		//destructor body
		}

3.	it can either accept arguments or not.	it can't have any arguments.
4.	A constructor is called when an instance or object of a class is created.	It is called while the scope of objects or program is over .
5.	Constructor is used to allocate the memory to an instance or object.	it is used to deallocate the memory of an object of a class.
6.	Constructor can be overloaded.	it can't be overloaded.
7.	The constructor's name is same as the class name.	Here, its name is also same as the class name preceded by the tiled (~) symbol.
8.	In a class, there can be multiple constructors.	While in a class, there is always a single destructor.
9.	Concept of Copy constructor is exist .	there is no concept of copy destructor.
10.	They are often called in successive order.	They are often called in reverse order of constructor.

Que : 4 What is operator overloading? Explain in detail.

- The mechanism of giving such special meanings to an operator is known as operator overloading.
- C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.
- For example '+' operator is used for arithmetic operation as well as concatenation of two strings.
- Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

Example:

```
 int a;
float b,sum;
sum=a+b;
```

Here, variables "a" and "b" are of types "int" and "float", which are built-in data types. Hence the addition operator '+' can easily add the contents of "a" and "b". This is because the addition operator "+" is predefined to add variables of built-in data type only.

Now, consider another example

```
class A
{
};
int main()
{
    A   a1,a2,a3;
    a3= a1 + a2;
    return 0;
}
```

In this example, we have 3 variables "a1", "a2" and "a3" of type "class A". Here we are trying to add two objects "a1" and "a2", which are of user-defined type i.e. of type "class A" using the "+" operator. This is not allowed, because the addition operator "+" is predefined to operate only on built-in data types. But here, "class A" is a user-defined type, so the compiler generates an error. This is where the concept of "Operator overloading" comes in.

Rules for operator overloading

1. Only existing operator can be overloaded. New operator cannot be created.
2. The overloaded operator must have at least one operand that is of user-defined type.
3. We cannot change the basic meaning of an operator. That is to say, we cannot redefine the plust(+) operator to subtract one value from the other.
4. Overloaded operator follows the syntax rules of the original operator.
5. Some operators cannot be overloaded such as.

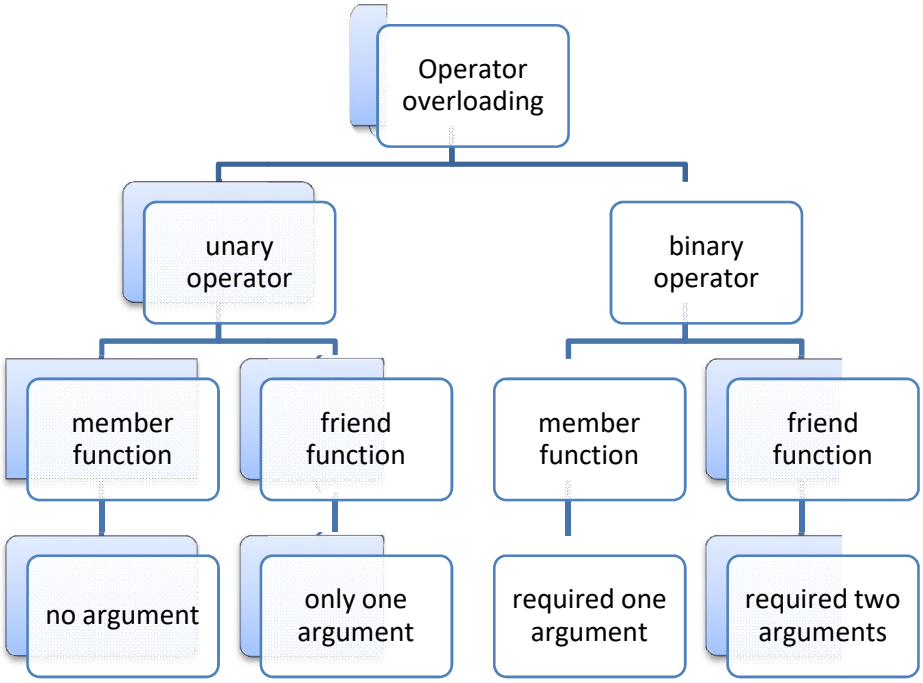
Operator	Name of Operator
sizeof	Sizeof operator
.	Membership operator
.*	Pointer to member operator
::	Scope resolution operator
?:	Conditional operator

Syntax:

```
return type classname::operator op(arglist)

{

        //Function body

}
```

Where return type can be float,int,char or class type,operator is a keyword which is used to overload an operator,op is the symbol of operator which we want to overload.

Types of operator Overloading



[1] Unary operator

An operator which required single operand to perform it is called unary operator.

Example:

Following are the examples of Unary operators:

- Unary minus (–) operator
- Logical not (!) operator
- Decrement (--) and Increment (++) operator

We can implement unary operator in two different ways such as

A. Member function
B. Friend function

[A]Implement using Member function:

- It must have operand as object.
- It does not require any argument.

Example

```
#include<iostream.h>
#include<conio.h>

class prefix
{
        int x,y;
        public:
                void get()
                {
                        cout<<"Enter value of x:";
                        cin>>x;

                        cout<<"\nEnter value of y:";
                        cin>>y;
                }

                void operator++()
                {
                        x=++x;
                        y=++y;
                }
                void put()
                {
                        cout<<"value of x:"<<x;
                        cout<<"\nValue of Y:"<<y;
                }
};
```

```
void main()
{
        clrscr();
        prefix p;
        p.get();
        ++p; //activates operator++() function
        p.put();
        getch();
}
```

[B]implement using Friend Function

- It must have operand as object.
- It requires single argument to implement it, and argument must be type of reference of object.
- Members must be define in public access specifier.
- Friend function cannot overload following operators.
    - Assignment operator(=)
    - Function call operator()
    - Subscript operator[]
    - Class member access operator(->)

Syntax:

```
Friend return_type operator op(class_name &obj)
{
    //function body
}
```

```
#include<iostream.h>
#include<conio.h>
class postfix
{
        int x,y;
        public:
                void get()
                {
                        cout<<"Enter value of x:";
                        cin>>x;

                        cout<<"\nEnter value of y:";
                        cin>>y;
                }


            friend void  operator++(postfix &p)
                {

                        p.x++;
                        p.y++;

                }
                friend void operator--(postfix &p)
                {

                        --p.x;
                        --p.y;
                                        }
                void put()
                {
                        cout<<"value of x:"<<x;
                        cout<<"\nValue of Y:"<<y;
                }
};
```

```
void main()
{
        clrscr();
        postfix p;
        p.get();
        ++p;  //operator ++ function activates
        p.put();

        p--; //operator -- function activates
        p.put();
        getch();
}
```

[2]Implement using Binary Operator

An operator which contains two operands to perform a mathematical operation is called the Binary Operator Overloading. It is a polymorphic compile technique where a single operator can perform various functionalities by taking two operands from the programmer or user.

It is used to perform operation on complex numbers such as object.

Example:

Following are the examples of binary operators:

- Assignment operator(=)
- Logical operator(&&,||)
- Relational Operator (>,<,>=,<=,!=)
- Bitwise operator(&,|)

We can implement binary operator in two different ways such as

a) Member function
b) Friend function

[A]implements using member function

- Required only single argument to perform it.
- Must be defining in public access specifier.
- Operands must be as complex type.

Example

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class String
{
        Char str[20];
        public:
                void get()
                {
                        Cout<<"Enter string:";
                        Cin>>str;
                }


             void operator+(String s1)
                {
                        strcat(str,s1.str)
                }

                void put()
                {
                        cout<<"string :"<<str;
                }
};
void main()
{
        clrscr();
        String s1,s2;
        s1.get();
        s2.get();

        s1+s2;
        s1.put();
        getch();
}
```

[B]implements using friend function

- Required two arguments to perform it.
- Arguments must be type of objects
- Members must be defining in public access specifier.
- Operands must be as complex type.

Example

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class String
{
        Char str[20];
        public:
                void get()
                {
                        Cout<<"Enter string:";
                        Cin>>str;
                }


            friend String operator+(String &s1,String s2)
                {
                        strcat(s1.str,s2.str)
                        return s1;
                }

                void put()
                {
                        cout<<"string :"<<str;
                }
};
```

```
void main()
{
        clrscr();
        String s1,s2,s3;
        s1.get();
        s2.get();

        s3=s1+s2;
        s3.put();
        getch();
}
```