

# CS-112

# Object Oriented Programming

## (3+1)

## Prerequisites:

## Programming Fundamentals

**SYED MUHAMMAD RAFI**

LECTURER, DEPARTMENT OF SOFTWARE ENGINEERING  
FACULTY OF ENGINEERING SCIENCE AND TECHNOLOGY,  
ZIAUDDIN UNIVERSITY

# Operator Overloading

**Lec-7**

# Operator Overloading

- The process of adding additional meanings of operator is known as **operator overloading**.
- It enables an operator to perform different operations depending on the type of operands.

- The basic arithmetic operators such as  $+$ ,  $-$ ,  $*$ ,  $/$  normally works with basic types such as int , float etc.
- The application of these operators with basic types are defined in the language.
- However an error occur if these operators are used with user defined objects.

# Overloading an operator

- An operator can be overloaded by declaring a special member function in the class.
- The member function uses the keyword **operator** with the symbol of operator to be overloaded.

## Syntax

The syntax of overloading an operator is as follows:

```
return_type operator op()  
{  
    Function body;  
}
```

Where.,

return\_type      It indicates the type of value returned by the function.

Operator      It is the keyword that indicates that the member function is used to overload an operator.

Op      It is the symbol of operator to be overloaded.

## Example

Void operator ++()

```
{  
function body ;  
}
```

# Overloading Unary Operators

- A type of operator that works with single operand is called **unary operand**.
- The unary operators are overloaded to increase their capabilities.
- The unary operator that can be overloaded in C++ are as follows:

+   -   \*   !   ~   &  
++   --   ()   ->   new   delete

# Overloading ++ Operator

- The increment operator ++ is a unary operator.
- It works with single operand .
- It increase the value of operand by 1.
- It only works with numerical values by default.
- It can be overloaded to enable it to increase the values of data members of an object in same way.



❑ Write a program that overloads increment operator to work with user-defined objects

```
#include <iostream>
using namespace std;
```

```
class Count
```

```
{
```

```
private:
```

```
    int n;
```

```
public:
```

```
    Count()
```

```
    {
```

```
        n=0;
```

```
    }
```

```
    void show()
```

```
    {
```

```
        cout<<"n="<<n<<endl;
```

```
    }
```

```
void operator ++()
```

```
{
```

```
    n=n+1;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    Count obj;
```

```
    obj.show();
```

```
    ++ obj;
```

```
    obj.show();
```

```
}
```

```
n=0
n=1
```

```
Process returned 0 (0x0)   execution time : 0.489 s
Press any key to continue.
```

## How Above Program Work

- The above program overloads the increment operator ++ to work with the objects of all user-defined class count.
- It increases the value of data member n by 1.
- The user can use the same format to increase the value of object as used with integers.
- The above overloading only works in prefix notation.

## Operator Overloading with Returned Value

- The increment operator can be used in assignment statement to store the incremented value in another variable.
- Suppose a and b are two integers. The following statement will increment the value of a by 1 and then assign the new value to b:

`b = ++a;`

- The above functionality can be assigned to the operator by returning the new value from member function.
- The returned value can be stored in another object of same type on the left side of the assignment operator.

objects.

The overloaded function should return an object after incrementing the data

```
#include <iostream>
```

```
using namespace std;
```

```
class count
```

```
{
```

```
private:
```

```
int n;
```

```
public:
```

```
count()
```

```
{
```

```
    n=0;
```

```
}
```

```
void show()
```

```
{
```

```
    cout<<"n="<<n<<endl;
```

```
}
```

```
count operator ++()
```

```
{
```

```
    count temp;
```

```
    n=n+1;
```

```
    temp.n=n;
```

```
    return temp;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    count x,y;
```

```
    x.show();
```

```
    y.show();
```

```
    y=++x;
```

```
    x.show();
```

```
    y.show();
```

```
    return 0;
```

```
}
```

```
n=0  
n=0  
n=1  
n=1
```


```
Process returned 0 (0x0)   execution time : 0.032 s  
Press any key to continue.
```

## How Above Program Work

- The above program overloads increment operator.
- The member function increments the value of data member n.
- It stores the increment value in temporary object and returns the object.
- It allows the user to use the increment operator in an assignment operator.

# Overloading Postfix Increment Operator

- The increment operator works in two notations i.e prefix and postfix notation.
- The operator has to be overloaded separately for both notations.

 Write a program that overloads postfix increment operator to work with user defined objects.

```
#include <iostream>
```

```
using namespace std;
```

```
class count
```

```
{
    private:
        int n;
    public:
        count()
        {
            n=0;
        }
        void show()
        {
            cout<<"n ="<<n<<endl;
        }
        count operator ++()
        {
            count temp;
            n=n+1;
            temp.n=n;
            return temp;
        }
}
```

```
count operator ++(int)
```

```
{
    count temp;
    n=n+1;
    temp.n=n;
    return temp;
};
int main()
{
    count x;
    x.show();
    ++x;
    x++;
    x.show();
    return 0;
}
```

```
n =0
n =2
```

```
Process returned 0 (0x0)   execution time : 0.195 s
Press any key to continue.
```

## How Above Program Work

- The above program overloads increments operator for both prefix and postfix notation .
- The keyword **int** in following statement indicates that operator is overloaded for postfix:

Count operator ++(int)

- The use of **int** in parenthesis is not an integer parameter.
- It is simply a flag to compiler that indicates that the operator is overloaded for postfix notation.



# Overloading Binary Operator

- A type of operator that works with two operands is called binary operator.
- The binary operators are overloaded to increase their capabilities.

❏ Write a program that overloads postfix increment operator to work with user defined objects.

```
#include <iostream>
using namespace std;
class Add
{
private:
int a,b;
public:
Add()
{
a=b=0;
}
void in()
{
cout<<"Enter a:";
cin>>a;
cout<<"Enter b:";
cin>>b;
}
void show()
{
cout<<"a ="<<a<<endl;
cout<<"b ="<<b<<endl;
}
```

```
Add operator +(Add p)
{
Add temp;

temp.a =p.a +a;

temp.b =p.b +b;

return temp;
};
int main()
{
Add x,y,z;

x.in();

y.in();


z=x+y;

x.show();

y.show();

z.show();

return 0;
}
```



```
Enter a:2
Enter b:4
Enter a:5
Enter b:7
a =2
b =4
a =5
b =7
a =7
b =11
```

## How Above Program Work

- The above program overloads the binary operator.
- When the binary object is used with objects, the left operand acts as calling object and right operand acts as parameter passed to the function.
- $z=x+y;$
- In the above statement,  $x$  is the calling object and  $y$  is passed as parameter.
- The member function adds the value of calling object and parameter object.
- It stores the the result in a temporary object and the return the temporary object.
- The return object is copied to the right side of assignment operator in `main()`



End of lecture