اسم المادة : برمجة الحاسوب
اسم التدريسي : زيد ابراهيم رسول
المرحلة : الثانية
السنة الدراسية :2024-2023
عنوان المحاضرة: oop

Ministry of Higher Education and Scientific Research

Al-Mustaqbal University

Computer Engineering Techniques Department

C/C++ programming language

Prepared By

MSc. Zaid Ibrahim Rasool

E-mail: zaid.ibrahim@uomus.edu.iq

Operator Overloading

Operator overloading is one of the most exciting features of object oriented programming. It can transform complex, obscure program listings into intuitively obvious ones. For example, statements like

```
d3.add_distances(d1, d2);
```

or the similar

```
d3 = d1.add_distances(d2);
```

can be changed to the much more readable

```
d3 = d1 + d2;
```

The term *operator overloading* refers to giving the normal C++ operators, such as +, *, <=, and +=, additional meanings when they are applied to user-defined data types like classes.

Defining Operator Overloading

To define an additional task to an operator, we must specify its means in relation to the class to which the operator is applied. This is done by using the ***operator*** *function*. The general form of an operator function is:

E-mail: zaid.ibrahim@uomus.edu.iq

```
return_type          classname :: operator op(arglist)
{
Function body                          // task defined
}
```

where :

return_type : is the type of value returned by the specified operation

op : is the operator being overloaded

"operator *op***"** : is the function name

Overloading Unary Operators

Unary operators act on only one operand. Examples of unary operators are the increment and decrement operators ++ and --, and the unary minus, as in -33. The unary minus when applied to an object should change the sign of each of its data items. The following example will show how to overload a unary operator.

Example: Write a C++ program using **operator overloading** to increment the value of **X, Y** and **Z** by one then make the negative values of **X, Y** and **Z**.

```
#include <iostream>
Using namespace std;
Class space
{
```

```
        private:
                int x;
                int y;
                int z;
        public:
               void getdata(int a, int b, int c);
               void display()
                   {
                   cout << x <<" "<< y <<" "<< z <<"\n";
                   }

Void operator++();     //overload increment
Void operator-();    //overload unary minus
};
Void space :: getdata(int a, int b, int c)
{
   x = a; y = b; z = c;
}
Void space :: operator++()
{
   x++;
   y++;
   z++; }
```

E-mail: zaid.ibrahim@uomus.edu.iq

```
void space :: operator-()
{
    x = -x;
    y = -y;
    z = -z;
}

void main()
{
    Space S;
    S.getdata(10, -20, 30);

    cout<<"S : ";
    S.display();
    ++S;                // activates operator++() function
   cout<<"S : ";
    S.display();

    -S;                 // activates operator-() function
    cout<<"S : ";
    S.display();
}
```

The output of the above program is

S : 10 -20 30
S : 11 -19 31
S : -11 19 -31

Note that a statement like S2 = ++S1;

will **not** work because the function operator++() does not return any value. It can work if the function is modified to return an object.

Overloading Binary Operators

Binary operators act on two operands. Examples include +, -, *, and /. The mechanism of overloading binary operators is the same as that of unary operators. In the program below we will see how to add two complex numbers using binary operator overloading.

Example: Write a C++ program using object and classes to sum any two complex numbers using operator overloading.

```
#include <iostream>
Using namespace std;
Class complex
{
        private:
                float x;
                float y;
```

```
    public:
            complex(){ }
            complex(float real, float imag)
                { x = real;    y = imag; }
             complexoperator+(complex);
             voiddisplay();
};


Complex complex :: operator+(complex c)
       {
   Complex temp;
   temp.x = x + c.x;
   temp.y = y + c.y;
   return temp;
       }
void complex :: display()
   { cout<< x << " + j"<< y << "\n"; }
void main()
       {
   Complex C1(2.5, 3.5), C2(1.6, 2.7),
   C3; C3 = C1 + C2;
```

```
cout<<"C1 = "; C1.display();
cout<<"C2 = "; C2.display();
cout<<"C3 = ";  C3.display();
     }
```

The output of the program above is

C1 = 2.5 + j3.5 C2 = 1.6 + j2.7 C3 = 4.1 + j6.2

Note that, in overloading of binary operators, the left-hand operand is used to invoke the operator function and the right-hand operand is passed as an argument.