UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA
DEPARTMENT OF COMPUTER SCIENCE
http://web.uettaxila.edu.pk

CS-103 | Object Oriented Programming (Lab)

LAB Manual 10

Operator Overloading

Instructor

Muhammad Faheem Saleem

Overloaded constructor

We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading and is quite similar to function overloading

- Overloaded constructors essentially have the same name (exact name of the class) and differ by number and type of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

C++ program to illustrate:

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

class construct
{

public:
    float area;

    // Constructor with no parameters
    construct()
    {
        area = 0;
    }

    // Constructor with two parameters
    construct(int a, int b)
    {
        area = a * b;
    }

    void disp()
    {
        cout<< area<< endl;
    }
};
```

```
int main()  {
   // Constructor Overloading with two different constructors of class name
construct o;
construct o2( 10, 20);

o.disp();
o2.disp();
return 0;
}
```

Output:

```
0  200
```

Operator Overloading

Operator overloading allows operators to be redefined and used with user-defined data types. It helps in performing operations on objects just like primitive data types. C++ can provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can make use of the addition operator (+) for string class to concatenate two strings. We know that the task of this operator is to add two operands. So a single operator '+', when placed between integer operands, adds them and when placed between string operands, concatenates them.

Operator overloading allows you to redefine the behavior of operators for user-defined types. This means you can use operators with objects of your own classes and define how they should behave when used with those objects.

For example, you can define how the + operator should work for objects of a Vector class to perform vector addition, or how the << operator should work to output objects of a custom Fraction class. Operator overloading is a compile-time polymorphism

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 let's consider another example to Demonstrate the working/Logic behind Operator Overloading

```
class A {
statements;
};
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.

Now, if the user wants to make the operator "+" add two class objects, the user has to redefine the meaning of the "+" operator such that it adds two class objects. This is done by using the concept of "Operator overloading". So the main idea behind "Operator overloading" is to use C++ operators with class variables or class objects. Redefining the meaning of operators really does not change their original meaning; instead, they have been given additional meaning along with their existing ones. Forexample:

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


 class Complex {

private:
 int real, imag;

public:

   Complex(int r = 0, int i = 0)
 {        real = r;
          imag = i;
}
   // This is automatically called when '+' is used with between two Complex objects

 Complex operator+(Complex const& obj)
    {
       Complex res;        res.real =
real + obj.real;        res.imag =
imag + obj.imag;
       return res;
    }
   void print() { cout << real << " + i" << imag << '\n'; }
};
 int main()
{    Complex c1(10, 5), c2(2, 4);
Complex c3 = c1 + c2;     c3.print();
}
```

Output

```
12 + i9
```

The Complex class represents complex numbers with two private member variables: real for the real part and imag for the imaginary part. The class has a constructor Complex(int r = 0, int i = 0) which initializes the real and imaginary parts of the complex number. It has default parameter values of 0. The operator+ function is overloaded inside the Complex class to perform addition of two complex numbers. It takes a reference to a Complex object (obj) as a parameter and returns a Complex object. Inside the operator+ function, it creates a new Complex object res, adds the real parts and imaginary parts of the two complex numbers (real + obj.real and imag + obj.imag), and then returns the result. The print function prints the complex number in the form "real + iimag". In the main() function, two Complex objects c1 and c2 are created with values (10, 5) and (2, 4) respectively. The + operator is overloaded for the Complex class, so c1 + c2 is equivalent to c1. operator+(c2). The result of the addition is stored in c3.Finally, the print function is called on c3 to print the result.

Types of Operator Overloading

Operator Overloading is the method by which we can change the function of some specific operators to do some different tasks.

Syntax:

```
Return_Type classname :: operator op(Argument list)
{
  Function Body
}   // This can be done by declaring the function
```

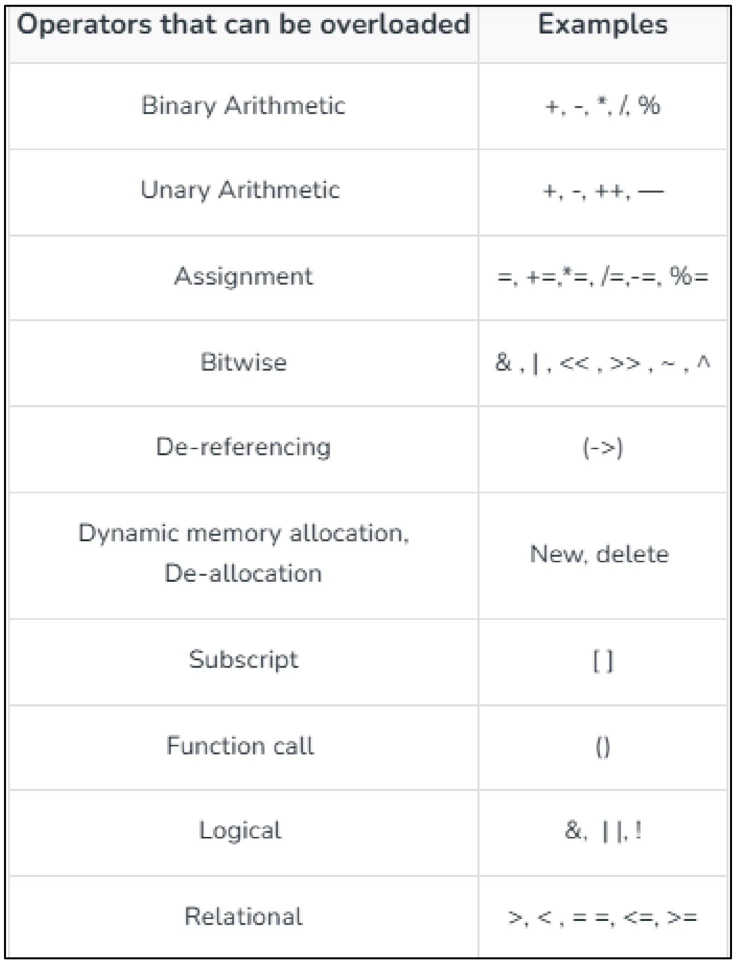
Here,

Return_Type is the value type to be returned to another object. **operator op** is the function where the operator is a keyword.

op is the operator to be overloaded.

Operator overloading in C++ can be classified into different types based on the number of operands they operate on. Here we discuss two types as

- Overloading Unary Operator.
- Overloading Binary Operator.



Operators that can be overloaded	Examples
Binary Arithmetic	+, -, *, /, %
Unary Arithmetic	+, -, ++, —
Assignment	=, +=,*=, /=,-=, %=
Bitwise	& , | , << , >> , ~ , ^
De-referencing	(->)
Dynamic memory allocation, De-allocation	New, delete
Subscript	[ ]
Function call	()
Logical	&, | |, !
Relational	>, < , = =, <=, >=

Binary Operator Overloading:

Binary operators are those that operate on two operands. Examples include +, -, *, /, ==, !=, etc. Binary operator overloading involves defining the behavior of these operators when used with objects of a class

In the binary operator overloading function, there should be one argument to be passed. It is the overloading of an operator operating on two operands. Below is the C++ program to show the overloading of the binary operator (+) using a class Distance with two distant objects.

```
// C++ program to show binary operator overloading
#include <iostream>
using namespace std;
class Distance { public:
```

```
int feet, inch;

    Distance()
    {        this->feet = 0;
             this->inch = 0;
    }

    Distance(int f, int i)
    {        this->feet = f;
             this->inch = i;
    }

    // Overloading (+) operator to perform addition of two distance object Call by reference
Distance operator+(Distance& d2)
    {
        // Create an object to return
        Distance d3;



        d3.feet = this->feet + d2.feet;
       d3.inch = this->inch + d2.inch;

        // Return the resulting object
return d3;
    }
};

int main()
{
    Distance d1(8, 9);
    Distance d2(10, 2);
    Distance d3;

    // Use overloaded operator
d3 = d1 + d2;

    cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;
    return 0;
}
```

Explanation:

Distance operator+(Distance &d2): Here return type of function is distance and it uses call by references to pass an argument.

d3 = d1 + d2: Here, d1 calls the operator function of its class object and takes d2 as a parameter, by which the operator function returns the object and the result will reflect in the d3 object.

Example: Overloading `+` and `-` Operators for a Complex Number Class

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

class Complex {
private:
    int real, imag;
public:
    Complex(int r = 0, int i = 0) : real(r), imag(i) {}

    Complex operator+(const Complex &obj) {
        return Complex(real + obj.real, imag + obj.imag);
    }

    Complex operator-(const Complex &obj) {
        return Complex(real - obj.real, imag - obj.imag);
    }

    void display() {
        cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complex c1(3, 4), c2(1, 2);
    Complex c3 = c1 + c2;
    Complex c4 = c1 - c2;

    cout << "Sum: ";
    c3.display();
    cout << "Difference: ";
    c4.display();

    return 0;
}
```

This C++ program demonstrates operator overloading for a Complex class, which represents complex numbers. The class has two private members, real and imag, to store the real and imaginary parts of a complex number. The + and - operators are overloaded to perform addition and subtraction of two complex numbers, respectively. The display() function prints the complex number in the format a + bi. In the main() function, two complex numbers c1 and c2 are created, and their sum (c3) and difference (c4) are calculated using the overloaded operators. Finally, the results are displayed, showing the sum and difference of the complex numbers

Unary Operator Overloading

Unary operators are those that operate on a single operand. Examples include ++, --, +, -, !, etc. Unary operator overloading involves defining the behavior of these operators when applied to objects of a class.

Example :

Let us consider overloading (-) unary operator. In the unary operator function, no arguments should be passed. It works only with one class object. It is the overloading of an operator operating on a single operand. Assume that class Distance takes two member objects i.e. feet and inches, and creates a function by which the Distance object should decrement the value of feet and inches by 1 (having a single operand of Distance Type).

```
#include <iostream>
```

```
using namespace std;

class Distance {
 public:
    int feet, inch;

    // Constructor to initialize the object's value
    Distance(int f, int i)
    {        this->feet = f;
             this->inch = i;
    }

    // Overloading(-) operator to perform decrement operation of Distance object
void operator-()
    {
feet--;
inch--;



 cout << "\nFeet & Inches(Decrement): " <<
              feet << "'" << inch;
    }
};



int main()
{
    Distance d1(8, 9);

    // Use (-) unary operator by single operand
    -d1;
return 0; }
```

Output

```
Feet & Inches(Decrement): 7'8
```

Explanation: In the above program, it shows that no argument is passed and no return_type value is returned, because the unary operator works on a single operand. (-) operator changes the functionality to its member function.

Overloading Relational Operators

Example: Overloading == Operator

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

class Box {
private:
    int length;
public:
    Box(int l)
{ length = l; }
    bool operator==(Box b) {
        return length == b.length;
    }
};

int main() {
    Box b1(5), b2(5);
    if (b1 == b2) // Checks if two boxes have the same length
        cout << "Boxes are equal" << endl;
    else
```

```
        cout << "Boxes are not equal" << endl;
    return 0;
}
```

The operator== function compares the lengths of two Box objects.

Example : Overloading `==`, `<`, and `>` for a Student Class

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

class Student {
private:
    int marks;
public:
    Student(int m) : marks(m) {}

    bool operator==(const Student &obj) {
        return marks == obj.marks;
    }

    bool operator<(const Student &obj) {
        return marks < obj.marks;
    }

    bool operator>(const Student &obj) {
        return marks > obj.marks;
    }
};
int main() {
    Student s1(85), s2(90);
    if (s1 == s2)
        cout << "Both students have equal marks." << endl;
    else if (s1 < s2)
        cout << "Student 1 has fewer marks." << endl;
    else
        cout << "Student 1 has more marks." << endl;

    return 0;
}
```

Lab Tasks

- **Write a program to overload binary operator + to find sum of two complex numbers using friend function.**
- **Develop a `Box` class and overload `<` and `>` operators to compare box volumes.**
- **Suppose there is a class called X with two double type attributes. Write a C++ program to create two objects named ob 1 and ob 2 of the above class and overload the binary == operator to perform the following operation within main():**

  ```
  if(ob 1== ob 2)
  cout<<"Objects are same"<<endl; else
  cout<<"Objects are different"<<endl;
  ```