

# Operator Overloading

Ali Haider

[syedalihaider.ciit@gmail.com](mailto:syedalihaider.ciit@gmail.com)

Department of Computer Science IUB

# Overview

- Operator Overloading
- Overloadable and Non Overloadable Operators
- Unary Operator Overloading
- Binary Operator Overloading
- Relation Operator Overloading

# Operator Overloading

- You can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.
- Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

# Overloadable Operators

+	-	*	/	%	^
&		~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	
+=	-=	/=	%=	^=	&=
=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

# Non Overloadable Operators

..

+

.

?

# Unary Operators

- The unary operators operate on a single operand and following are the examples of Unary operators –
  - The increment (++) and decrement (--) operators.
  - The unary minus (-) operator.
  - The logical not (!) operator.
- The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object,
- As in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

# Increment (++) Overloading

```
#include <iostream>
using namespace std;
class Distance {
private:
    int feet;           // 0 to infinite
    int inches;         // 0 to 12
public:
    // required constructors
    Distance() {
        feet = 0;
        inches = 0;
    }
    Distance(int f, int i) {
        feet = f;
        inches = i;
    }
    // method to display distance
    void displayDistance() { cout << "F: " << feet << " I:" << inches << endl; }
    // overloaded Increment (++) operator
    Distance operator++ () {
        feet = ++feet;
        inches = ++inches;
        return Distance(feet, inches);
    }
};
```

1

```
int main() {
    Distance D1(11, 10), D2(-5, 11);

    ++D1;           // apply increment
    D1.displayDistance(); // display D1

    ++D2;           // apply increment
    D2.displayDistance(); // display D2

    return 0;
}
```

2

# Binary Operators

- The binary operators take two arguments and following are the examples of Binary operators.
- You use binary operators very frequently like addition (+) operator, subtraction (-) operator and division (/) operator.

# Binary Operators + and - Overloading

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

class Box {
private:
    double length;    // Length of a box
    double breadth;    // Breadth of a box
    double height;    // Height of a box
public:
    double getVolume(void) {
        return length * breadth * height;
    }
    void setLength( double len ) {
        length = len;
    }
    void setBreadth( double bre ) {
        breadth = bre;
    }
    void setHeight( double hei ) {
        height = hei;
    }
};
```

1

```
// Overload + operator to add two Box objects.
Box operator+(const Box& b) {
    Box box;
    box.length = this->length + b.length;
    box.breadth = this->breadth + b.breadth;
    box.height = this->height + b.height;
    return box;
}

// Overload - operator to subtract two Box objects.
Box operator-(const Box& b) {
    Box box;
    box.length = this->length - b.length;
    box.breadth = this->breadth - b.breadth;
    box.height = this->height - b.height;
    return box;
}

};
```

2

Cont...

```
int main() {  
    Box Box1;           // Declare Box1 of type Box  
    Box Box2;           // Declare Box2 of type Box  
    Box Box3;           // Declare Box3 of type Box  
    double volume = 0.0; // Store the volume of a box here  
    // box 1 specification  
    Box1.setLength(6.0);  
    Box1.setBreadth(7.0);  
    Box1.setHeight(5.0);  
    // box 2 specification  
    Box2.setLength(12.0);  
    Box2.setBreadth(13.0);  
    Box2.setHeight(10.0);  
    // volume of box 1  
    volume = Box1.getVolume();  
    cout << "Volume of Box1 : " << volume << endl;  
    // volume of box 2  
    volume = Box2.getVolume();  
    cout << "Volume of Box2 : " << volume << endl;  
    // Add two object as follows:  
    Box3 = Box1 + Box2;  
    // volume of box 3  
    volume = Box3.getVolume();  
    cout << "Volume of Box3 after Addition : " << volume << endl;  
    // Subtract two object as follows:  
    Box3 = Box1 - Box2;  
    // volume of box 3  
    volume = Box3.getVolume();  
    cout << "Volume of Box3 after Subtraction : " << volume << endl;  
    return 0;  
}
```

# Relational Operators

- There are various relational operators supported by C++ language like (<, >, <=, >=, ==, etc.) which can be used to compare C++ built-in data types.
- You can overload any of these operators, which can be used to compare the objects of a class.
- Following example explains how a < operator can be overloaded and similar way you can overload other relational operators.

# Relational Operators < Overloading

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

class Distance {
private:
    int feet;           // 0 to infinite
    int inches;         // 0 to 12
public:
    // required constructors
    Distance() {
        feet = 0;
        inches = 0;
    }
    Distance(int f, int i) {
        feet = f;
        inches = i;
    }
    // method to display distance
    void displayDistance() {
        cout << "F: " << feet << " I:" << inches << endl;
    }
};
```

1

```
// overloaded < operator
bool operator <(const Distance& d) {
    if(feet < d.feet) {
        return true;
    }
    if(feet == d.feet && inches < d.inches) {
        return true;
    }
    return false;
};

int main() {
    Distance D1(11, 10), D2(5, 11);
    if( D1 < D2 ) {
        cout << "D1 is less than D2 " << endl;
    } else {
        cout << "D2 is less than D1 " << endl;
    }
    return 0;
}
```

2