S. S Jain Subodh P.G. (Autonomous) College

SUBJECT - Object Oriented Programming Concept

TITLE - Operator Overloading

Created By:
Shalu J. Rajawat

Introduction

- Operator overloading
  - Enabling C++'s operators to work with class objects
  - Using traditional operators with user-defined objects
  - Requires great care; when overloading is misused, program difficult to understand
  - Examples of already overloaded operators
    - Operator **<<** is both the stream-insertion operator and the bitwise left-shift operator
    - **+** and **–**, perform arithmetic on multiple types
  - Compiler generates the appropriate code based on the manner in which the operator is used

- Overloading an operator
  - Write function definition as normal
  - Function name is keyword **`operator`** followed by the symbol for the operator being overloaded
  - **`operator+`** used to overload the addition operator (**+**)
- Using operators
  - To use an operator on a class object it must be overloaded unless the assignment operator **(=)** or the address operator **(&)**
    - Assignment operator by default performs memberwise assignment
    - Address operator (&) by default returns the address of an object

Restrictions on Operator Overloading

- C++ operators that can be overloaded

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

- C++ Operators that cannot be overloaded

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Unary operators overloading in C++

.

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--.

Following example explain how minus (-) operator can be overloaded for prefix as well as postfix usage.

```
#include <iostream.h>
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;
                }
                void displayDistance() {// method to display distance
                        cout << "F: " << feet << " I:" << inches <<endl;
                }
                // overloaded minus (-) operator
                Distance operator- () {
                        feet = -feet;
                        inches = -inches;
                        return Distance(feet, inches);
                }
};
```

```
int main()
 {
            Distance D1(11, 10), D2(-5, 11);
            -D1; // apply negation
           D1.displayDistance(); // display D1
           -D2; // apply negation
           D2.displayDistance(); // display D2
           return 0;
 }
```

OUTPUT:

F: -11 I:-10

F: 5 I:-11

Binary operators overloading in C++

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.

Following example explains how addition (+) operator can be overloaded. Similar way, you can overload subtraction (-) and division (/) operators.

```
#include <iostream.h>
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;
                }
                 // 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;
                }
```

```
private:
                double length; // Length of a box
                double breadth; // Breadth of a box
                double height; // Height of a box
};


 // Main function for the program
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 : " << volume <<endl;
        return 0;
 }
```

OUTPUT:

Volume of Box1 : 210
Volume of Box2 : 1560
Volume of Box3 : 5400

12

Relational operators overloading in C++

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.

```
#include <iostream.h>
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 minus (-) operator
                Distance operator- () {
                        feet = -feet;
                        inches = -inches;
                         return Distance(feet, inches);
                }
```

```
                // 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;
}
```

OUTPUT:

D2 is less than D1

Overloading Increment ++ and Decrement --

The increment (++) and decrement (--) operators are two important unary operators available in C++.

Following example explain how increment (++) operator can be overloaded for prefix as well as postfix usage. Similar way, you can overload operator (--).

```
#include <iostream.h>
class Time {
        private:
                int hours; // 0 to 23
                int minutes; // 0 to 59
        public:
                // required constructors
                Time(){
                        hours = 0; minutes = 0;
                }
                Time(int h, int m){
                        hours = h; minutes = m;
                }
                // method to display time
                void displayTime() {
                        cout << "H: " << hours << " M:" << minutes <<endl;
                }
```

```
// overloaded prefix ++ operator
Time operator++ () {
        ++minutes; // increment this object
        if(minutes >= 60) {
                ++hours;
                minutes -= 60;
        }
        return Time(hours, minutes);
}
// overloaded postfix ++ operator
Time operator++( int ) {
        // save the orignal value
        Time T(hours, minutes);
         // increment this object
        ++minutes;
        if(minutes >= 60) {
                 ++hours;
                minutes -= 60;
         }
         // return old original value
        return T;
 }
};
```

```
int main() {
        Time T1(11, 59), T2(10,40);
        ++T1; // increment T1
        T1.displayTime(); // display T1
        ++T1; // increment T1 again
        T1.displayTime(); // display T1
        T2++; // increment T2
        T2.displayTime(); // display T2
        T2++; // increment T2 again
        T2.displayTime(); // display T2
        return 0;
    }
```

OUTPUT:

H: 12 M:0
H: 12 M:1
H: 10 M:41
H: 10 M:42

THANKS