Practical Lecture : Operator Overloading 1

Quick Recap

Let’s take a quick recap of previous lecture –

A)

B)

C)

D)

E)

Today's Agenda

Today we are going to cover -

- What is Operator Overloading?
- Syntax
- Can all operator overloaded?
- Types of operator overloading
- Rules for operator overloading
- Different Approaches to operator overloading

Let’s Get Started-

Operator Overloading

In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.

Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.

Syntax

```
class class_name
{
   ... .. ...
   public
     return_type operator symbol (argument(s))
     {
        ... .. ...
     }
   ... .. ...
};
```

- The return_type is the return type for the function.
- Next, you mention the operator keyword.
- The symbol denotes the operator symbol to be overloaded. For example, +, -, <, ++.
- The argument(s) can be passed to the operator function in the same way as functions.

Example

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

class TestClass {
private:
    int count;
public:
    TestClass() : count(5) {}
    void operator --() {
        count = count - 3;
    }
    void Display() {

        cout << "Count: " << count; }
};
```

Example

```
int main() {
    TestClass tc;
    --tc;
    tc.Display();
    return 0;
}
```

Can all operator be overloaded?

No. There are C++ operators that can't be overloaded.

They include:

- :: -Scope resolution operator
- ?: -ternary operator.
- . -member selector
- Sizeof operator
- * -member pointer selector

Types of operator overloading

- Unary operator overloading
- Binary operator overloading

Different approach to operator overloading

Operator Overloading can be done by using **three approaches**, they are

- Overloading unary operator.
- Overloading binary operator.
- Overloading binary operator using a friend function.

Rules For operator overloading

- In case of a non-static function, the binary operator should have only one argument and unary should not have an argument.
- In the case of a friend function, the binary operator should have only two argument and unary should have only one argument.
- All the class member object should be public if operator overloading is implemented.
- Operators that cannot be overloaded are **. .* :: ?:**
- Operator cannot be used to overload when declaring that function as friend function **= () [] ->**.

Unary Operator Overloading Example

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

Unary Operator Overloading Example

```
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;
      }
```

Unary Operator Overloading Example

```
// 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);
    }
};
```

Unary Operator Overloading Example

```
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 Operator Overloading Example

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

Binary Operator Overloading Example

```
class Box {
   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;
   }
```

Binary Operator Overloading Example

```
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;
}
```

Binary Operator Overloading Example

```
  // 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);
```

Binary Operator Overloading Example

```
  // 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

QNA Time

Any Questions ??

Any Questions??

Thank You!

See you guys in next class.