OPERATOR OVERLOADING

With operator overloading , a programmer is allowed to provide his own definition for an operator to a class by overloading the built in operator . This enables the programmer to perform specific computation when the operator is applied on class objects and apply a standard definition when the same operator is applied on a built in data type.

> Operator Overloading is also a form of compile time polymorphism.

Advantages :

Operator overloading enables programmers to use notation closer to the target domain. Example for adding two matrices , we can simply write M1 + M2 , rather than M1.add(M2).

Operator Overloading makes programs look clear.

EXAMPLE CODE FOR OPERATOR OVERLOADING :

```
Complex operator +(Complex &c2)
{
        Complex Temp;
        Temp.real = real + c2.real;
        Temp.imag = imag + c2.imag;
        Return Temp;
}
```

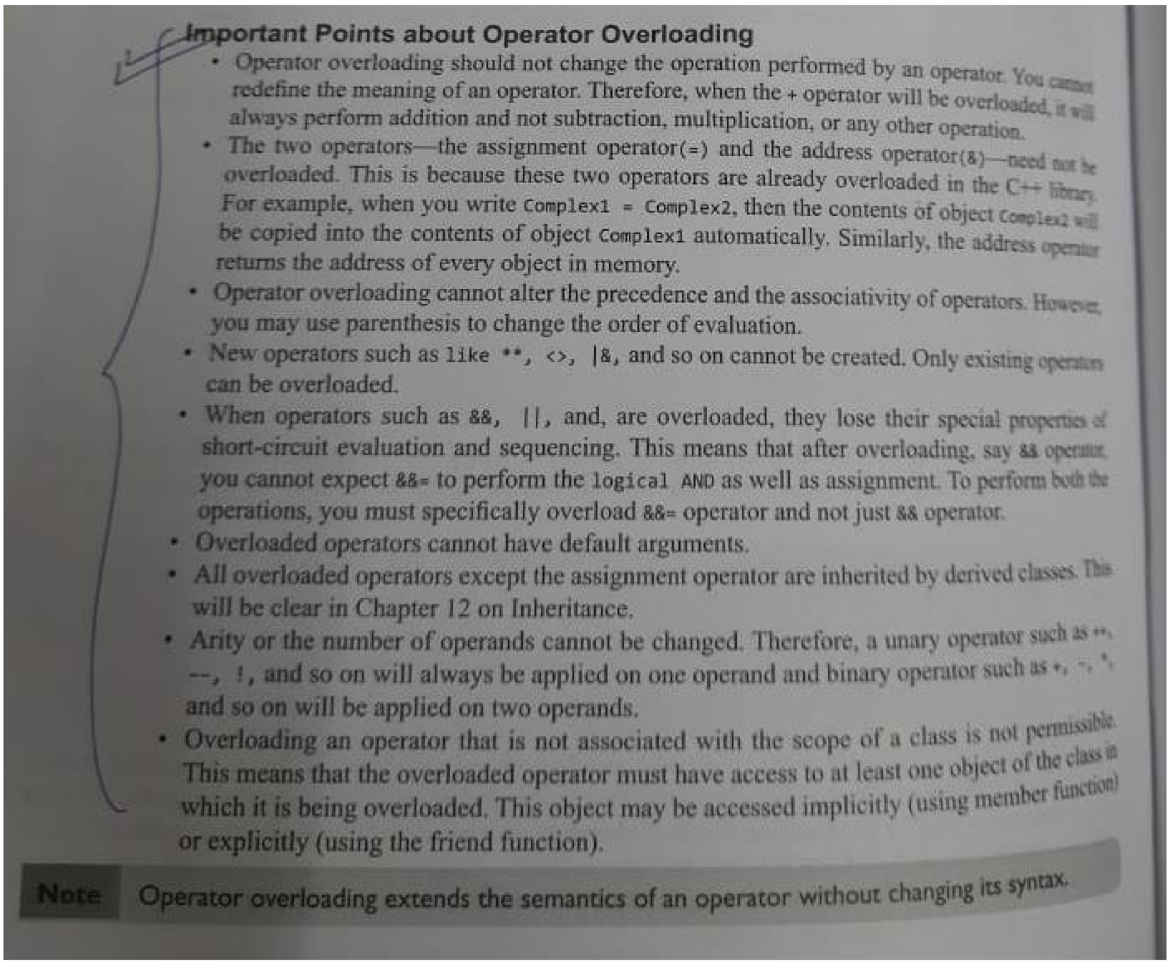
OPERATORS THAT CAN BE OVERLOADED :

! , ++ , -- , + , - , * , / , % , ^ , = , == , += , != , -= , & , | , && , || , *= , /= , %= , ^= , &= , <<= , >>= , <<>>

<u>OPERATORS THAT CANNOT BE OVERLOADED</u> :

Scope Resolution operator (: :)
Member Selection Operator (.)
Member selection through a pointer to a function (.*)
Ternary operator (?:)
Size of operator (sizeof)

<u>IMPORTANT POINTS ABOUT OPERATOR OVERLOADING</u> :



Important Points about Operator Overloading

- Operator overloading should not change the operation performed by an operator. You cannot redefine the meaning of an operator. Therefore, when the + operator will be overloaded, it will always perform addition and not subtraction, multiplication, or any other operation.
- The two operators—the assignment operator(=) and the address operator(&)—need not be overloaded. This is because these two operators are already overloaded in the C++ library. For example, when you write `Complex1 = Complex2`, then the contents of object `Complex2` will be copied into the contents of object `Complex1` automatically. Similarly, the address operator returns the address of every object in memory.
- Operator overloading cannot alter the precedence and the associativity of operators. However, you may use parenthesis to change the order of evaluation.
- New operators such as `like **, <>, |&`, and so on cannot be created. Only existing operators can be overloaded.
- When operators such as `&&, ||`, and, are overloaded, they lose their special properties of short-circuit evaluation and sequencing. This means that after overloading, say `&&` operator, you cannot expect `&&=` to perform the `logical AND` as well as assignment. To perform both the operations, you must specifically overload `&&=` operator and not just `&&` operator.
- Overloaded operators cannot have default arguments.
- All overloaded operators except the assignment operator are inherited by derived classes. This will be clear in Chapter 12 on Inheritance.
- Arity or the number of operands cannot be changed. Therefore, a unary operator such as `++, --, !`, and so on will always be applied on one operand and binary operator such as `+, -, *`, and so on will be applied on two operands.
- Overloading an operator that is not associated with the scope of a class is not permissible. This means that the overloaded operator must have access to at least one object of the class in which it is being overloaded. This object may be accessed implicitly (using member function) or explicitly (using the friend function).

Note Operator overloading extends the semantics of an operator without changing its syntax.