CSC127 Operator Overloading

Operator Overloading

- “Operator overloading is the **ability** to **tell** the **compiler** how to perform a certain **operation** when its corresponding **operator** is used on one or more variables.”

- Operator overloading is **closely** related to **function overloading**.

- Allows the full **integration** of **new class types** into the programming environment.

Operator Overloading

- Overloading of operators are achieved by creating **operator function**.

- "An ***operator function*** defines the operations that the overloaded operator can perform relative to the class".

- An operator function is created using the keyword **operator.**

Operator Overloading

- Operator functions can be either **members** or **nonmembers** of a class.

- **Non-member** operator functions are always **friend functions** of the class.

Operator Overloading

- **Overloadable operators are:**

  + - * / = < > += -= *= /= << >>

  <<= >>= == != <= >= ++ -- % & ^

  ! | ~ &= ^= |= && || %= [] ()

  new delete

Operator Overloading

- **Creating a Member Operator Function**

Within Class:

ret-type operator#(argument-list);

Outside Class:

ret-type class-name::operator#(arg-list)

{

// operations

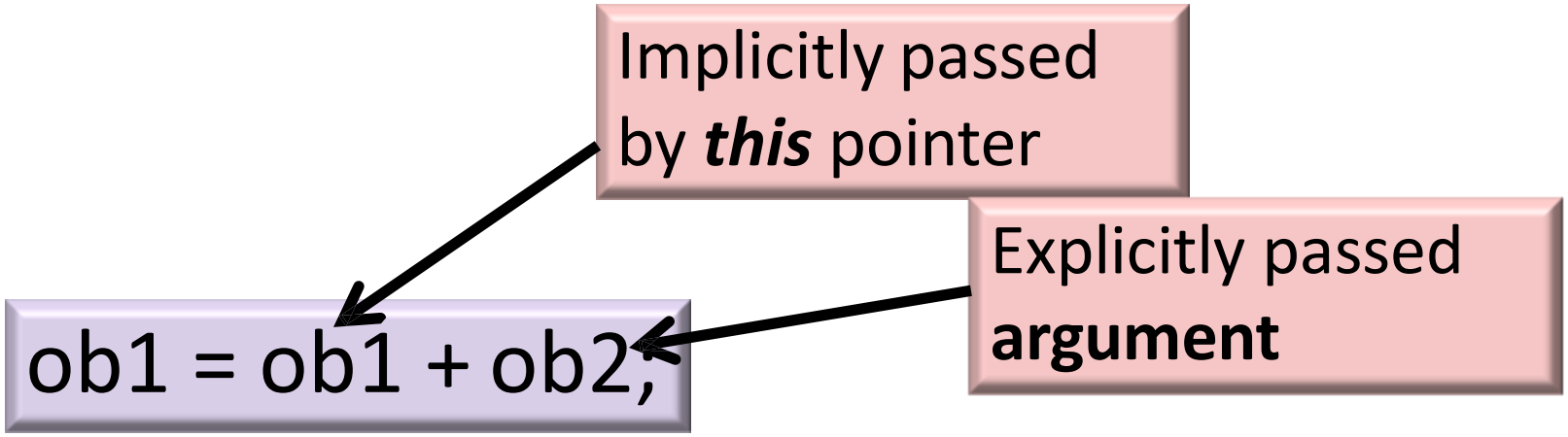
}

Operator Overloading

Example:

```
comp    comp::operator+(comp      op2  )
{
comp temp;
temp.real = op2.real + real;
temp.img = op2.img + img;
return temp;
}
```

Operator Overloading




```
ob1 = ob1.operator+ (ob2);
```

Operator Overloading Restrictions

- Should not alter the **precedence** of an operator.
- Should not change the **number of operands** that an operator takes.
- Operator functions cannot have **default arguments**.
- Operators cannot be overloaded are:

  . : : .* ?: sizeof

Why not . : : .* ?: sizeof() **operators?**

- The restriction is for safety. If we overload . operator then we cant access member in normal way.

- The :? takes 3 argument rather than 2 or 1. There is no mechanism available by which we can pass 3 parameter during operator overloading.

Why not . : : .* ?: sizeof() operators?

- Example:

  c = a+b - both a & b actually refer to some memory location, so "+" operator can be overloaded,

  but the "." operator, like a.i actually refers to the name of the variable from whom the memory location has to be resolved at time and thus it cannot be overloaded.