

Operator overloading

- Version 1: Dr. Ofir Pele
- Version 2: Dr. Erel Segal-Halevi

Operator Overloading - functions

- **Operators** like `+`, `-`, `*`, are actually **functions**, and can be overloaded.
- Can be overloaded as functions for existing classes (*folder 0*):

```
string operator* (string s, int n)
{
    . . .
}
```

Operator Overloading - methods

- Operators can be overloaded as **methods** too
(*folder 1*):

```
class Complex {  
    ▪    Complex operator* (Complex b)  
        {  
            . . .  
        }  
}
```

Note the different number of arguments – 2 vs. 1.
"this" is always the first argument.

Invoking an Overloaded Operator

Operator can be invoked as a member function:

```
object1.operator* (object2) ;
```

It can also be used in more conventional manner:

```
object1 * object2;
```

What is it good for

- Natural usage.
- compare:
 - **a.set(add(b,c))**
 - to
 - **a= b+c**
- compare:
 - **v.elementAt(i)= 3**
 - to
 - v[i]= 3**

Operators ++ -- postfix prefix (folder 1)

// Prefix: ++n

```
Complex& operator++() {  
    code that adds one to this  
    return *this; // return ref to curr  
}
```

A flag that makes
it postfix

// Postfix : n++

```
const Complex operator++(int) {  
    Complex cpy(*this); // calling copy ctor  
    code that adds one to this  
    return cpy;  
}
```

Arithmetic operators in the standard library

- plus (+) is used for concatenating strings.
- Arithmetic operators are used for `valarray` objects.
- NOTE: operators are not commutative!
 $a*b$ does not have to be the same as $b*a$.
 (mathematical example: `matrix*vector`)

Other operators in the standard library

- `>>` `<<` are used as bit operations for **primitives numbers** and for I/O in the **standard library** `iostreams` classes.
- `[]` is used as subscripting **primitives arrays** and vector class in the **standard library** (see folder 2).
- `()` is used for **function calls** and for functor objects in the **standard library** (see folder 4).

Rules for writing your operators

1. **Don't** overload operators with **non-standard** behavior! (<< for adding,...)
2. Check how operators work on **primitives** or in the **standard library** and give the **same behavior** in your class.