

Operator overloading part 1

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

Operator Overloading

- **Operators** like +, - , * , are actually **methods**,
- and can be overloaded.
- **Can be overloaded even for existing classes** (*folder 0*).

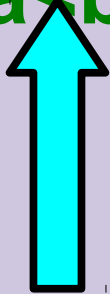
What is it good for - 1

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

What is it good for - 2

Uniformity with base types (important for templates)

```
template<typename T>  
const T& min(const T& a, const T& b) {  
    return a < b ? a : b;  
}
```



a and b can be primitives Or

user defined objects that have operator <

Complex example *(folder 1)*

Rules

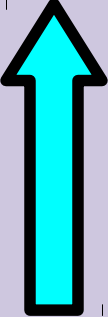
1. **Don't** overload operators with **non-standard** behavior! (<< for adding,...)
2. Check how operators work on **primitives** or in the **standard library** and give the **same behavior** in your class.

Example of usage in **primitives/standard library**

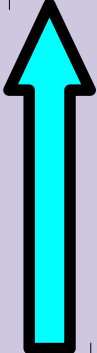
- `>>` `<<` are used as bit operations for **primitives numbers** and for I/O in the **standard library** `iostreams` classes.
- `[]` is used as subscripting **primitives arrays** and vector class in the **standard library**.
- `()` is used for **function calls** and for functor objects in the **standard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

Prototype

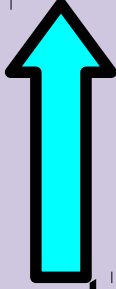
`X& operator+=(const X& rval)`



return
type



method
name



parameter for
object on right
side of operator

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;
```