Some Expressions Cannot Be Made Equivalent

Other equivalences **cannot be made consistent** if overloading is used to substantially change some operators.

For example, pointer dereference:

- `inst->member`
- `(*inst).member`
- `inst[0].member`

The latter two expressions use "`.`", which **cannot be overloaded**.

Assignment Requires First Destroying the Old Instance

Similarly, given

```
ALPHA a;
```

the **following are not equivalent**:

```
ALPHA b = a;  // copy constructor
b = a;    // operator= (assignment)
```

For a new variable, the first version is better:

- **work** may be **required to destroy** `b`
- (for example, removing `b` from a sorted list or a binary search tree).
- **before copying** the new value **from** `a`.

Copy Constructor and Assignment not Equivalent in C++

Note that

- the **copy constructor** and
- **assignment** (operator=)
- are **not equivalent in C++**.

The **default versions are the same** (copy constructor detailed in earlier slides), but

- overriding one does NOT override the other,
- which continues to use the default version
- and compilers will NOT warn you.

Treating Instances as C Variables Generates Useless Work

One last topic: variable declarations and single-assignment.

C++ instances are "always" valid:

- constructed when they are declared, and
- destructed when they leave scope (and can no longer be accessed).

Treating them as C variables generates useless work by forcing initialization before information for initialization is available.

