“Data Transfer Dilemma”

● How do you make some object (e.g., $x$) get another object’s (e.g., $y$’s) value?
● Before: $x = 7$ \hspace{1cm} $y = 13$

● After: $x =$ \hspace{1cm} $y =$

● So, what’s the problem?!!
“Data Transfer Dilemma”

- How do you make some object (e.g., $x$) get another object’s (e.g., $y$’s) value?
- Before: $x = 7$ \hspace{2cm} $y = 13$
- After: $x = 13$ \hspace{2cm} $y = 13$
- So, what’s the problem?!?
“Deep” vs. “Shallow” Copying

- **Before:** $x \quad y$
- **After:** $x \quad y$

- So, what’s the problem?!?
“Deep” vs. “Shallow” Copying

- Before: $x \rightarrow y$

- After: $x \rightarrow y$

- So, what’s the problem?!?
Effect Of Aliasing

- Loss of the modular (compositional) reasoning property!

- Example:

```c
void P (int& x, int& y);
    // ensures  x = #x + 1  and
    //           y = #y + 2
...
```

```c
a = 10;
P (a, a);
```
Analysis Of What “x = y;” Does

What does “x = y;” do?

- deep copy
- shallow copy

How big is y’s rep?

- small
- large
Analysis Of What “x = y;” Does

What does “x = y;” do?

- deep copy
- shallow copy

How big is y’s rep?

- small
- large

Yes
Analysis Of What “x = y;” Does

What does “x = y;” do?

- deep copy
- shallow copy

How big is y’s rep?

- small
- large

✓ ✓ ✓
Analysis Of What “x = y;” Does

What does “x = y;” do?
- deep copy
- shallow copy

How big is y’s rep?
- small
- large

- [✓]
- [✗]
Data Transfer Dilemma Revisited

- How do you make some object (e.g., $x$) get another object’s (e.g., $y$’s) value?

  - Before: $x = \quad y =$

  - After: $x = \quad y =$

  - What is $y$’s value after “transfer”?
Some Possibilities...

- $y$ is preserved (as noted, OK if $y$ is cheap to copy, but otherwise trouble)
- $y$ becomes undefined
- $y$ becomes an arbitrary legal value
- $y$ becomes old $x$ (i.e., swap $x$ and $y$)
Data Transfer Dilemma Revisited

How do you make some object (e.g., $x$) get another object’s (e.g., $y$’s) value?

Before: $x =$ \[\text{cow}\] $y =$ \[\text{horse}\]

After: $x =$ \[\text{horse}\] $y =$ \[\text{cow}\]

This is what happens with swapping, i.e., $x := y$
To Use Swapping, Must You Change The Way You Program?

Two answers:

- **No** — the only real change is when replacing an assignment of the form “\( x = y; \)”, where the next occurrence of \( y \) is a “use” (as opposed to a “definition”); an empirical question

- **Yes** — it changes the way you should design component interfaces
Example: A Queue Component

- Operations: Enqueue, Dequeue, Length
- What should Enqueue do to the item being enqueued?
  - Preserve it?
  - Change it? (If so, how?)
Significance

• All objects/variables can have value semantics, allowing modular reasoning
• All references/pointers are invisible, hidden deep within the bowels of a very few “lowest-level” components
• If these lowest-level components have no storage leaks, then client programs have no storage leaks