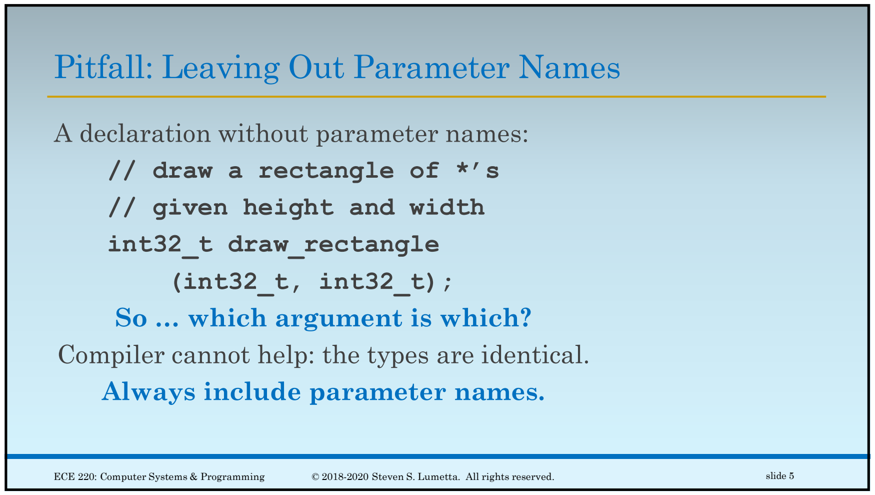


Pitfall: Leaving Out Parameter Names

A declaration without parameter names:

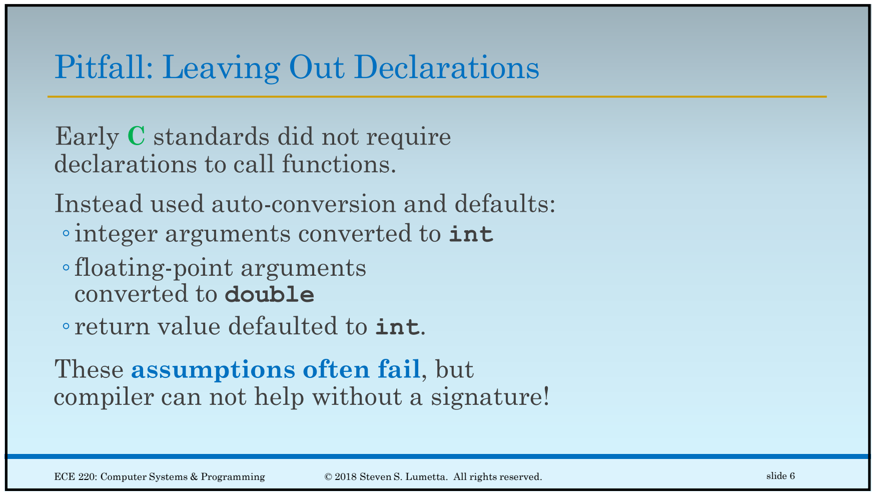
```
// draw a rectangle of *'s
// given height and width
int32_t draw_rectangle
        (int32_t, int32_t);
```

So … which argument is which?

Compiler cannot help: the types are identical.

Always include parameter names.

5



Pitfall: Leaving Out Declarations

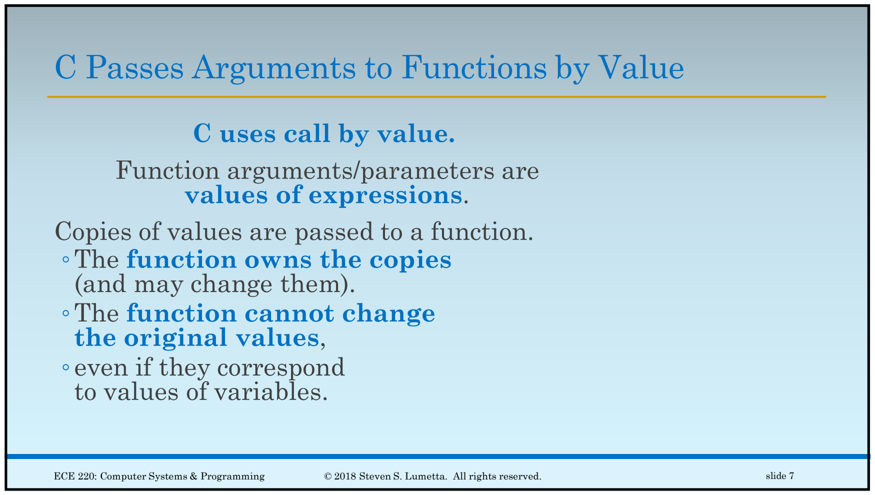
Early **C** standards did not require declarations to call functions.

Instead used auto-conversion and defaults:

- integer arguments converted to `int`
- floating-point arguments converted to `double`
- return value defaulted to `int`.

These **assumptions often fail**, but compiler can not help without a signature!

6



C Passes Arguments to Functions by Value

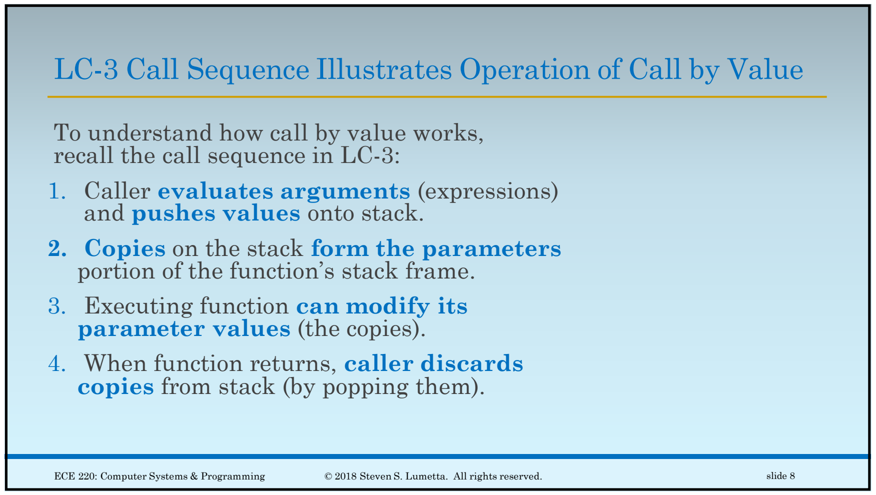
C uses call by value.

Function arguments/parameters are **values of expressions**.

Copies of values are passed to a function.

- The **function owns the copies** (and may change them).
- The **function cannot change the original values**,
- even if they correspond to values of variables.

7



LC-3 Call Sequence Illustrates Operation of Call by Value

To understand how call by value works, recall the call sequence in LC-3:

1. Caller **evaluates arguments** (expressions) and **pushes values** onto stack.
2. **Copies** on the stack **form the parameters** portion of the function's stack frame.
3. Executing function **can modify its parameter values** (the copies).
4. When function returns, **caller discards copies** from stack (by popping them).

8