

# CSE 333

## Mini-lecture 13 - revisiting references

**Hal Perkins**

Department of Computer Science & Engineering

University of Washington





# ∃ confusion about references

When should they be used?

- as arguments?
- as return values?

When can using them cause trouble?



# Let's go through examples

I'll show you some code, you tell me whether:

- (a) we must use a reference
- (b) it's OK and encouraged to use a reference
- (c) it's OK but discouraged to use a reference
- (d) we must NOT use a reference



see `arg1.cc`



# arg1.cc

- (a) we must use a reference
- (b) it's OK and encouraged to use a reference
- (c) it's OK but discouraged to use a reference**
- (d) we must NOT use a reference

For simple primitive types (int, float, etc.), passing in a const reference results in a correct program, but the performance benefit is questionable.



see [arg2.cc](#)



# arg2.cc

- (a) we must use a reference
- (b) it's OK and encouraged to use a reference**
- (c) it's OK but discouraged to use a reference
- (d) we must NOT use a reference

For complex types (structs, object instances), passing in a const reference results in a correct program and likely gives you some performance benefits.

- pop quiz: why not pass in a pointer instead?



see `ret1.cc`



# ret1.cc

- (a) we must use a reference
- (b) it's OK and encouraged to use a reference
- (c) it's OK but discouraged to use a reference
- (d) we must NOT use a reference**

Never return a reference to a local (stack allocated) variable; it's the same error as returning a pointer to one.



see `Complex1.h`



# Complex1.h

**(a) we must use a reference**

(b) it's OK and encouraged to use a reference

(c) it's OK but discouraged to use a reference

(d) we must NOT use a reference

A copy constructor must have a reference parameter (that identifies it as a copy ctr). `const` could be omitted but is almost always used. It is correct, safe, and efficient.



see `Complex2.h`



# Complex2.h

- (a) we must use a reference
- (b) it's OK and encouraged to use a reference
- (c) it's OK but discouraged to use a reference
- (d) we must NOT use a reference**

Because we don't want to return <a reference to \*this>, but instead <a copy of a local variable>, we cannot use a reference in this case.

- pop quiz: does chaining work if we correct the code?



see `Complex3.h`



# Complex3.h

**(a) we must use a reference**

(b) it's OK and encouraged to use a reference

(c) it's OK but discouraged to use a reference

(d) we must NOT use a reference

We must use a reference so chaining works correctly. It is also more efficient to use a reference.

- pop quiz: why does chaining break if we don't use a reference? give an example of chained code that breaks.



see `Complex4.h`



# Complex4.h

- (a) we must use a reference**
- (b) it's OK and encouraged to use a reference
- (c) it's OK but discouraged to use a reference
- (d) we must NOT use a reference

This is the same case as the plain assignment operator; we must return a reference so that chaining works.



see `Complex5.h`



# Complex5.h

- (a) we must use a reference**
- (b) it's OK and encouraged to use a reference
- (c) it's OK but discouraged to use a reference
- (d) we must NOT use a reference

This is the same case as the assignment operator; we must return a reference so that chaining works. More so, copying `std::cout` doesn't make sense (and is prevented)!



See you on Friday!