

N. Meng, S. Arthur

- Shallow binding?
- Deep binding?
- Ad hoc binding?

Referencing Environment for Subroutine Parameters

- Deep binding and ad hoc binding can be the same when a subroutine is declared and passed by the same subroutine
- In reality, ad hoc binding has never been used
- Static-scoped languages usually use deep binding

N. Meng, S. Arthur

 Dynamic-scoped languages usually use shallow binding

function Sent() { print(x); function Receiver(func) { var x; x = 2; function Sender() {

var x; x = 1: Receiver(Sent)

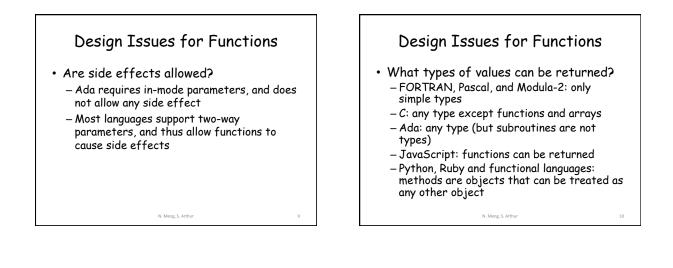
};

3:

An Example

 In static-scoped languages, Receiver is not always visible to Sent, so deep binding is natural

 In dynamic-scoped languages, it is natural for Sent to have access to variables in Receiver, so shallow binding is appropriate





• A subroutine that has the same name as another subroutine in the same referencing environment, but its number, order, or types of parameters must be different

– E.g., void fun(float); void fun():

 C++ and Ada have overloaded subroutines built-in, and users can write their own overloaded subroutines

N. Meng, S. Arthur



- A generic or polymorphic subroutine takes parameters of different types on different activations
- An example in C++ template<class Type> Type max(Type first, Type second) {
 return first > second ? first: second; int a, b, c; char d, e, f; c = max(a, b);N. Meng, S. Arthu f = max(d, e);

14

Generic Subroutine

- Overloaded subroutines provide a particular kind of polymorphism called ad hoc polymorphism
 - Overloaded subroutines need not behave similarly
- **Parametric polymorphism** is provided by a subroutine that takes generic parameters to describe the types of parameters

N. Meng, S. Arthur

• Parametric polymorphic subroutines are often called generic subroutines

Coroutine

- A special kind of subroutine. Rather than the master-slave relationship, the caller and callee coroutines are on a more equal basis
- A coroutine is a subroutine that has multiple entry points, which are controlled by coroutines themselves

N. Meng, S. Arthur

Coroutine

- The first execution of a coroutine begins at its beginning, but all subsequent executions often begin at points other than the beginning
- Therefore, the invocation of a coroutine is named a **resume**
- Typically, coroutines repeatedly resume each other, possibly forever
- Their executions interleave, but do not overlap

N. Meng, S. Arthur

	sub co1() {
An Example	 resume(co2);
The first time co1 is resumed, its execution begins at the first	 resume(co3); }
statement, and executes down to resume(co2) (with the statement included)	L
 The next time co1 is resumed, its execution begins at the first statement after resume(co2) 	
 The third time co1 is resumed, its execution begins at the first statement after resume(co3) 	
N. Meng, S. Arthur	16

Coroutine

- The interleaved execution sequence is related to the way multiprogramming operating systems work
 - Although there may be one processor, all of the executing programs in such a system appear to run concurrently while sharing the processor
 - This is called quasi-concurrency
- Coroutines provide quasi-concurrent execution of program units

N. Meng, S. Arthur