COMP 302 Lecture 7 26 September 2016

Bindings

- Explicitly created; x=1; y=2…
- Also happens at function call time
  - function foo(x) { …} foo(7) → x = 7

Scope: What bindings we can access and how

a: …
x: … this is the environment of foo

- Sets of bindings → environment
  - function foo (a) { var x; …}
- Scope" the textual area in which a (set of) bindings is defined.
- 2 main flavours of scope:
  - Static scoping: C, Java, JS
  - Dynamic scoping: early *LISP, SNOBIL, TeX*

Static scoping:

- "Lexical scoping" different name for the same thing
- Find a variable by (statically) examining the code
- Simplest model → one big global scope
  - early forms of *basic*
  - access every var from every var. Every time you access a given variable name, you always get the same value.
- "Slightly saner": dividing the scope into at least two pieces.
  - Global scope: access anywhere.
  - Local scope: only within a context (typically procedure, function…) *see 7c302*
  - Usually we allow nesting of local scopes
  - What we can access in nested scopes.
    - *Algol-style*: A name declared in a scope is known to all inner, nested scopes down to the point at which an identically-named variable is declared.
  - nb: JS is a bit different
    - *see 7c302-1*
- Does declaration order matter?
  - var x = 1; var y = x; //okay
  - var y = x; var x = 1; //should that be okay
  - Based on an old language *Modula-3*: order does not matter. e.g. in JS, *see 7c302-2*. This applies to functions
  - For variables, *see 7c302-3*

Dynamic scoping

- The bindings/environment chain that we could build just by looking at the code for static scoping changes at runtime. That is the difference. *see 7c302-4*

Static scoping
Global
n: 1
x: __
y: __
x and y point to it directly

Dynamic scoping
Global
n: 1
x: __
y: __
y points to it directly
x points to y then global

- Dynamic scoping makes it easy to make errors. But *see 7c302-5*

…

- Functions define scopes
  - *When* do we create a scope and assign its parent?
    - When you *call* the function, or
    - When you *refer* to the function.
- Late/shallow binding
  - This is the one we've been using. Late: when you call the function
- Early/deep binding
  - When you pass as argument the function. *see 7c302-6*