Administrative Notes

Extra classes on Monday at 11am in 373 Soda.

- 11am-12:30pm on Monday, February 26 and
- 11am-12:30pm on Monday, March 4

Strict vs. Lazy Evaluation (Review)

Lazy evaluation (call by name):

$d \in Denotable = Computation$

$$\mathcal{E}[\![I]\!] = \lambda e\ .\ (with\text{-}denotable(lookup e I)(\lambda d\ .\ d))$$
$$\begin{aligned}\mathcal{E}[\![(\texttt{call}\ E_1\ E_2)]\!] = \lambda\ e.(\ & with\text{-}procedure(\mathcal{E}[\![E_1]\!]e)\\ & (\lambda p.(p(\mathcal{E}[\![E_2]\!]e))))\end{aligned}$$

Eager evaluation (call by value):

$d \in Denotable = Value$

$$\mathcal{E}[\![I]\!] = \lambda e\ .\ (with\text{-}denotable(lookup e I)(\lambda d\ .\ (val\text{-}to\text{-}comp d)))$$
$$\begin{aligned}\mathcal{E}[\![(\texttt{call}\ E_1\ E_2)]\!] = \lambda\ e.(\ & with\text{-}procedure(\mathcal{E}[\![E_1]\!]e)\\ & (\lambda p.(with\text{-}value(\mathcal{E}[\![E_2]\!]e)p)))\end{aligned}$$

Affects the evaluation of identifies and function definitions.

Call by Denotation

$$d \in Denotable = Environment \rightarrow Computation$$

$$\begin{aligned}
\mathcal{E}[\![I]\!] &= \lambda e\ .\ (\mathit{with\text{-}denotable}(lookupeI)(\lambda d\ .\ (d\ e))) \\
\mathcal{E}[\![(\texttt{call}\ E_1\ E_2)]\!] &= \lambda\ e.(\ \mathit{with\text{-}procedure}(\mathcal{E}[\![E_1]\!]e) \\
&\qquad (\lambda p.(p\ \mathcal{E}[\![E_2]\!])))
\end{aligned}$$

Scoping vs. Parameter Passing

The difference between call by denotation and call by name is the scope in which free variables in the actual parameter expression are evaluated.

There are two separable scoping issues:

- Where (in what scope) are free variables in the function body evaluated?
- Where are the free variables in the actual parameter expressions evaluated?

Usually people are refering to the first of these two when they discuss scoping.

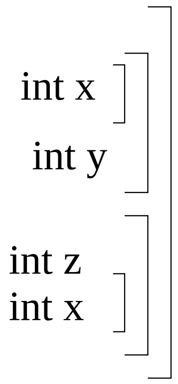
Scoping

Always look in innermost, current, scope first. If the variable isn't declared there, look in the closest enclosing scope. Where "closest" is defined by:

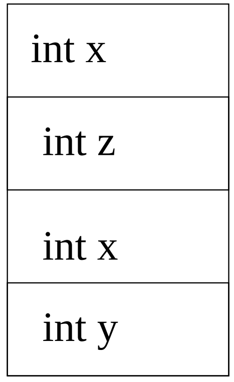
* Static scoping: in the hierarchical environment defined by the program structure

* Dynamically: in the nesting order defined by procedure invocations

Static



Dynamic



Dynamic Scoping

Dynamic scoping used in early Lisps, APL, and SNOBOL4.

Can be very handy for some problems.

```
(define derivative
   (lambda (f x)
     (/ (- (f (+ x epsilon))
           (f x))
        epsilon)))


(let ((epsilon 0.001))
 (derivative (lambda (x) (* x x)) 5.0))
```

Execution model seems simple–textual substitution of procedure body. But how simple are the semantics, if we want to think of procedures as mathematical functions?

Recall Funarg Problem (from Lecture 2)

Upward FUNARG: Lose bindings to F.V. because function returned as result.

```
((lambda x (lambda y (+ x y)))
```

static:

dynamic:

Downward FUNARG: (persistant binding) Function with F.V. passed as an argument.

```
((lambda x (lambda (f x) (f x)) (lambda y (+ x y)) 4) 1)
```

static:

dynamic

Denotational Semantics of Scoping

Static scoping:

$p \in Procedure = Denotable \rightarrow Computation$

$$\mathcal{E}[\![(\texttt{proc}\ I\ E)]\!] = \lambda e_{proc}\ .\ \begin{array}{l}(\textit{val-to-comp}(Procedure \mapsto Value \\ (\lambda d\ .\ (\mathcal{E}[\![E]\!][I\ :\ d]e_{proc}))))\end{array}$$

$$\mathcal{E}[\![(\texttt{call}\ E_1\ E_2)]\!] = \lambda\ e_{call}\ .\ (\ \begin{array}{l}\textit{with-procedure}(\mathcal{E}[\![E_1]\!]e_{call}) \\ (\lambda p.(p(\mathcal{E}[\![E_2]\!]e_{call}))))\end{array}$$

Dynamic scoping:

$p \in Procedure = Denotable \rightarrow Environment \rightarrow Computation$

$$\mathcal{E}[\![(\texttt{proc}\ I\ E)]\!] = \lambda e_{proc}\ .\ \begin{array}{l}(\textit{val-to-comp}(Procedure \mapsto Value \\ (\lambda\ x\ e_{call}\ .\ (\mathcal{E}[\![E]\!][I\ :\ d]e_{call}))))\end{array}$$

$$\mathcal{E}[\![(\texttt{call}\ E_1\ E_2)]\!] = \lambda\ e_{call}\ .\ (\ \begin{array}{l}\textit{with-procedure}(\mathcal{E}[\![E_1]\!]e_{call}) \\ (\lambda p.(p(\mathcal{E}[\![E_2]\!]e_{call})e_{call})))\end{array}$$

Environment Diagrams

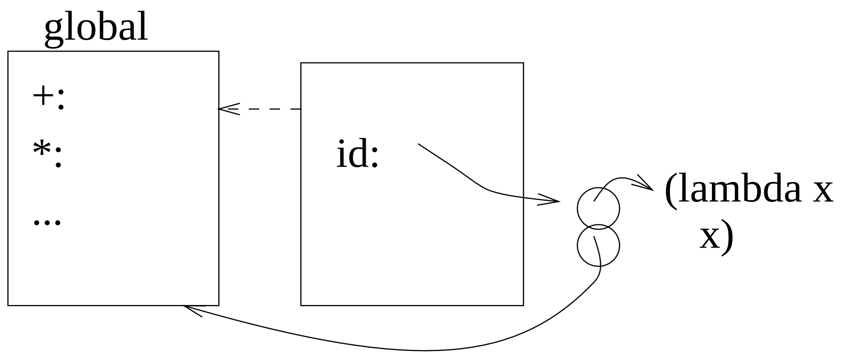
A procedure is represented by a closure containing an expression (code) and an environment.

A procedure activation is represented a local environment and two pointers:

static enclosing environment $\longrightarrow$

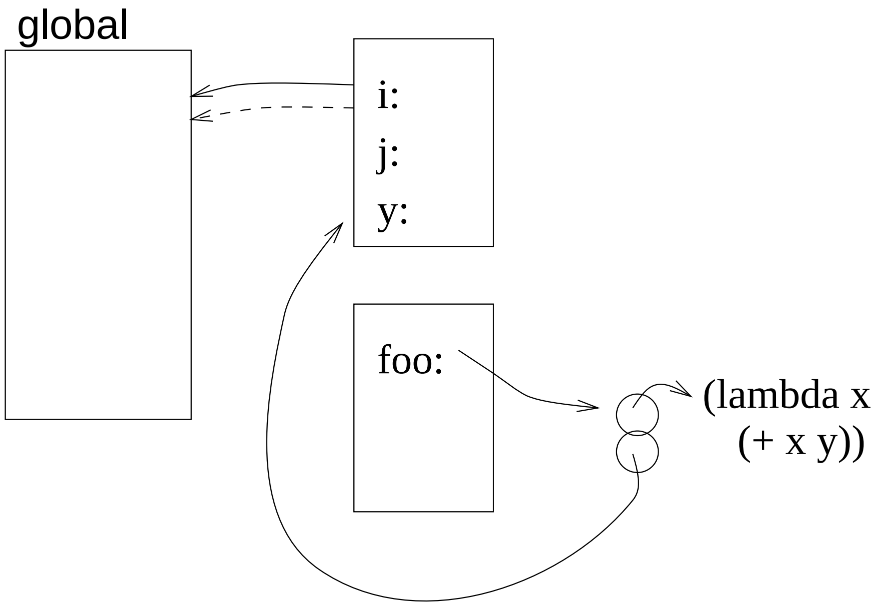
dynamic enclosing environment $- - - >$

(let ((id (lambda x x))) (id 3))



Environment Diagram Example

```
(let ((i 1) (j 2) (y 3))
     (let (foo (lambda (x) (+ x y)))
          (let ((i 4) (y 5))
               (foo (- i j)))))
```



Records in FL

Invented to decompose large programs, which do not fit into a flat naming discipline. A hierarchical name space (nested scopes) helps, but it not enough.

```
(record (I E)*)
```

Defines a binding between I's and E's, which make up an environment. Given a module value, expressed by E, we can get the value of an I by:

```
(select E I)
```

For example,

```
(let ((pack1 (module (val 2)
                     (scale (lambda (x) (+ x 10)))))
      (pack2 (module (vim 3)
                     (scale (lambda (x) (* x 2))))))
   ((select pack1 scale) (select pack2 vim)))
```

Records in FL

In addition, we can merge records and hide names using:

```
(override E1 E2)

(conceal (I*) E)
```

The `override` function creates a new records that has all of the fields of `E1` and `E2`, with the names in `E2` taking precedence if there are duplicates.

The `conceal` function creates a record that is just like `E`, but all of the names in `I*` are hidden.

What about these contructs is ambiguous?

Denotational Semantics of Records

$Value = ... + Record$
$Record = Environment$

$with\text{-}record : Computation \rightarrow (Record \rightarrow Computation) \rightarrow \; Computation$
usual definition

$$\begin{aligned}\mathcal{E}[\![(\texttt{record}\ (I_1\ E_1)...(I_n\ E_n))]\!] = \lambda e\ .\ \ & (with\text{-}values\ \mathcal{E}[\![E_1]\!]...\mathcal{E}[\![E_1]\!] \\ & \quad (\lambda\ v_1...v_n\ .\ val\text{-}to\text{-}comp\ (Record \mapsto Value \\ & \qquad (extend - env\ I_1\ v_1\ ... \\ & \qquad (extend_e nv\ I_n\ v_n\ empty\text{-}env)))))\end{aligned}$$

$$\begin{aligned}\mathcal{E}[\![(\texttt{select}\ I\ E)]\!] = \lambda\ e\ .\ (\ & with\text{-}record(\mathcal{E}[\![E]\!]e) \\ & (\lambda v\ .\ (with\text{-}denotable\ (lookup\ v\ I) \\ & \quad \lambda d\ .\ (val\text{-}to\text{-}comp\ d))))\end{aligned}$$

Records with Sugar (Modules)

```
(module (I E)*)

(with (I*) E1 E2)

(project (I*) E)

(rename ((I_old I_new)*)
```