LEXICAL ADDRESSING

Scope

```
> (define x                    ; x1
    (lambda (x)                ; x2
      (map
        (lambda (x)            ; x3
          (+ x 1))             ; ref x3
        x)))                   ; ref x2
> (x '(1 2 3))                 ; ref x1
(2 3 4)
```

Identifiers

- Identifiers refer to things such as variables or procedures.
- We try to use intuitive identifier names to make our lives easier.
- The computer does not care about names
- Compilers tend to rename all identifiers to generic names such as `Name001, Name002`
- In Scheme, due to lexical scope, we could get away without identifiers.

Lexical Addressing

- In Scheme, variable bindings can be determined by looking at the code.
- In other words, variable names are bound statically.
- We could refer to a variable by a number indicating the *position* in the parameter list of the lambda expression where the variable is bound.
- To deal with nesting, we identify the appropriate lambda expression by a second number which we call *lexical depth.*
- Lexical depth is zero based and indicates how many levels we need to go up from the current binding to get the binding we wish to refer to.
- As such, we can reference variables by the following pattern:

(<variable name>: <lexical depth> <position>)

Example of 1st Step in Lexical Addressing

- For example:

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

- Would become:

```
(lambda (x)
  (lambda (y)
    (+ (x : 1 0) (y : 0 0))))
```

Class demo

```
(lambda (x y)
  ((lambda (a)
     (x (a y)))
   x)
```

- We replace every variable reference *v* in an expression by:
  (v: d p) where *d* is the lexical depth and *p* is its declaration position.
- The above example becomes:

```
(lambda (x y)
  ((lambda (a)
     ((x : 1 0) ((a : 0 0) (y: 1 1))))
   (x : 0 0))
```

Lexical Addressing: Step 2

- Next, we completely eliminate variable names.
- Instead of a list of parameters, we simply write down a number, indicating how many variables got bound.
- For example

  `(lambda (x y z) …)`

  would become

  `(lambda 3 …)`

Lexical Addressing: Step 2

- Next, we remove the variable names from the variable reference expressions.
- We eliminate *v* from (v: d p), resulting in: (: d p)
- Example:

```
(lambda (x y)
  ((lambda (a)
     ((x : 1 0) ((a : 0 0) (y: 1 1))))
   (x : 0 0))
```

- Turns into:

```
(lambda 2
  ((lambda 1
     ((: 1 0) ((: 0 0) (: 1 1))))
   (: 0 0))
```

Class Exercises

- What is wrong with the following lexical address?

```
(lambda 1
   (lambda 1
      (+ (: 0 1) (: 1 0))))
```

- Convert the following into Scheme:

```
(lambda 2
  (lambda 1
     (+ (: 1 0) (: 1 1) (: 0 0))))
```