6.090, Building Programming Experience

Ben Vandiver

Lecture 3: More Procedures

Refer to solutions to homework 2 on the server. See solution for the "define even?" problem.

```
(define m 2)
((lambda (m) (* m m)) 5)
```

m is used as a valid name before it is used as a variable; so in the above evaluation, m does not represent the number 2; it represents 5.

Refer to solutions to homework 2 again. See solution for the evaluation of yummy.

`(lambda (   ) 5)`   this is called a thunk

No input variables

`((lambda (   ) 5))`

Combination involving a compound procedure. Evaluates to 5

Stepper

> M-s, gives stepper-top-level
> Hit space consecutively to see order of evaluation in scheme
> C-x, k kill buffer
> C-x, 1 one screen instead of split screen
>
> Scheme does not work with substitution…i.e., it does not use global name/value table.
>
> <u>N | V</u>
>
> It builds smaller fact name/value tables. If it does not find the binding here, it *then* goes to the global name/value table.

<u>name | value</u>

Special form: begin

Evaluate each expression in order and return the value of the last expression. The use of this special form is getting the return value desired with "side effects" that are also desired. These side effects will not affect the outcome of the return.

```
(begin
 3
 4
 5)
;Value: 5
```

```
(begin
 (display “yay”)
 5)
yay
;Value: 5
```

```
(begin
 (display “yay”)
 (display “rah”)
 5)
yahrah
;Value: 5
```

The values are returned in order as entered. If a lambda function were created instead for the same purpose, the values may be returned in different order.

```
((lambda (x y z) z)
 (display “yay”)
 (display “rah”)
 5)
rahyay
;Value: 5
```

Tower of Hanoi

Set of monks came up with a game of three rods and thirty-two rings.
The object of the game is for all rings to be moved over to the rod beside the beginning rod. One of the rods is where the rings begin, and the other is where they must all end up. The final one serves as a scratch…it is the swap space.

A bigger ring may not be placed on a smaller one. It is to be assumed that you can solve a problem of a smaller size. This can be assumed, because you initially solve this smaller problem and then you move on to the larger problem with the assumption.

(see lec3.scm on course website)
M-o evaluates all of the expressions in the file *.scm
The move command created in lec3.scm moves disc 1 from peg 1 to peg 2. If there are n discs, it will take around 2^(n-1) moves to complete.

Towers of Hanoi solution (n discs)
Base case: n=1 disc → (move 1 from to)
Recursive case: move n-1 discs from source to scratch, move the nth disc from source to destination, then move n-1 discs from scratch to destination.

Can move n-1 discs by recursively applying the Towers of Hanoi solution.

The function Hanoi may be defined in this way, after hitting M-o in lec3.scm. This way, it is possible to employ the move function without creating it.

```
(define hanoi
  (lambda (n from to scratch)
    (if (= n 1)
        (move n from to)
        (begin
         (hanoi (- n 1) from scratch to)
         (move n from to)
            (hanoi (- n 1) scratch to from)))))
```

Semi-colon: “comment character” scheme ignores everything written on lines preceded by semi-colons.

Homework
Do problems on Lecture 3 handout. Solutions are posted under Lecture 3 on the server.