Topic 7
More Hierarchical Data

Exercises 1.41, 1.42
Section 2.2.1
Section 2.2.2

October 2008

Back to Chapter 1

- Exercise 1.41
- Define a procedure double that takes a procedure of one argument as argument and returns a procedure that applies the original procedure twice.
- ((double inc) 1)
- 3

Another Chapter 1 Exercise

- Exercise 1.42
- Let f and g be two one-argument functions. The composition of f after g is defined to be the function

  x -> f(g(x)).

  Define a procedure compose that implements composition. For example:

  ((compose square inc) 6)

Functions that take any number of Arguments

- Some procedures in scheme (e.g., +, *, list) take an arbitrary number of arguments?
- How do we do that??
- Use define with *dotted-tail* notation
- End parameter list with a . before the last element. The parameters before the . get bound normally. The final formal parameter gets bound to the list containing the remaining actual parameters.

Dotted-Tail Notation in Define

- *(define (f x y . z) <body>)*
- *(f 1 2 3 4 5 6)*
- *x=1*
- *y=2*
- *z=(3 4 5 6)*
- *(define (g . w) <body>)*
- *(g 1 2 3 4 5)*
- *w= (1 2 3 4 5)*

Exercise 2.20

- Define a function same-parity that takes one or more integers and returns a list of all the arguments that have the same even-ldd parity as the first argument.

(same-parity 1 2 3 4 5 6 7)
(1 3 5 7)

(same-parity 2 3 4 5 6 7)
(2 4 6)

A more general append

```
Use the dotted tail notation to write a
  function that acts like append.  Call it
  gappend.  gappend takes any number of lists
  and returns a single list whose elements are
  the elements of the individual lists.

(gappend '(a b) '(c d) '(e f) '(g h))
(a b c d e f g h)
```

Spring 2008 Programming Development Techniques 7

Mapping over lists

```
Define a function square-list
```
; takes a list of numbers and returns a list containing
; the squares of the numbers in the original list

(square-list '(1 2 3 4))
(1 4 9 16)

Spring 2008 Programming Development Techniques 8

Mapping over lists

```
Define a function double-eles
```
; takes a list and returns a list containing
; the elements of the original list doubled
; in individual sublists

(double-eles '(a b c d))
((a a) (b b) (c c) (d d))

Spring 2008 Programming Development Techniques 9

Can this be generalized?

- Write a map procedure – and then define the earlier two procedures using map

Spring 2008 Programming Development Techniques 10

Scheme's map procedure

```
(map <procedure taking N arguments>
     <list 1>
     <list 2>
     ...
     <list N>)
```
All lists must have the same length.

Spring 2008 Programming Development Techniques 11

Using `map`

```
(define (vector-sum vec1 vec2)
  (map + vec1 vec2))

(vector-sum (list 1 2 3 4)
            (list 5 6 7 8))
--> (6 8 10 12)
```

Spring 2008 Programming Development Techniques 12

Arbitrarily Complex Lists

- Also called trees in the text – we have worked with these in a couple of procedures earlier –
- We write emb-subst and ?? (perhaps it was add-nums?)

Counting leaves

```
; takes a tree and returns the number
; of leaves in that tree
```

(count-leaves '(a ((b c) 2) (((e))) 7))
6

Multiplying all leaves by the same number – scale-tree

; takes a tree whose leaves are numbers
; and a number
; returns a similar tree with the numbers
; multiplied by num

```
if x --> ((2 1) (4 3)), then
(scale-tree x 5) --> ((10 5) (20 15))
```


A definition of scale-tree using `map`

(better than book's definition; works on one-leaf tree)

Solution to exercise 2.32

```
; generates the set of subsets of a set s

(subsets (list 1 2 3) -->
(() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3))
```