Worksheet #3
Michael Collins

Draw box and pointer diagrams for the following expressions:

(cons 1 2)
(list 1 2)
(cons 1 (cons 2 nil))
(cons 1 nil)
(cons 1 (cons 2 3))
(cons (cons 1 2) nil)
(list 1 2 3 4)
(list 1 (cons 2 3) (list 4 5))

In scheme, we will often want to access elements deep in a cons structure. Therefore, the following accessors have been designed to help us out:

(cadr x) = (car (cdr x))
(caddr x) = (car (cdr (cdr x)))
(cdaar x) = (cdr (car (cdr x)))
(cddr x) = (cdr (cdr x))

etc. etc.

say we define:

(define x (list 1 2 3 4 5 6 7 8 9 10))

How could you get the values 1, 2, 3, 4 using the c???r functions on x?

ANSWER:

(car x)
(cadr x)
(caddr x)
(caddr x)

Here’s a procedure which appends two lists together:

(define (append a b)
(if (null? a)
  b
  (cons (car a) (append (cdr a) b))))

Trace what happens (using box and pointer diagrams), when we apply
(append (list 1 2) (list 3 4 5))

Write a procedure that sums the elements in a list. The procedure
should generate a recursive process.

ANSWER:
(define sum (lambda (x)
  (if (null? x)
      0
      (+ (car x) (sum (cdr x))))))

Write a procedure "square-ls", that takes a list as input, and returns
a new list with all values within the list squared. For example
(square-ls (list (1 2 3 4))) => (1 4 9 16)

ANSWER:
(define (square-ls x)
  (if (null? x)
      nil
      (cons (* (car x) (car x)) (square-ls (cdr x)))))

Write a procedure that sums the elements in a list. The procedure
should generate an iterative process.

ANSWER:
(define sum (lambda (x)
  (sum-h 0 x)))

(define sum-h (lambda (p x)
  (if (null? x)
      p
      (sum-h (+ p (car x)) (cdr x)))))

Take the following definition for a procedure "foo":

2
(define foo (lambda (x)
    (foo-h x nil)))

(define foo-h (lambda (x y)
    (if (null? x)
        y
        (foo-h (cdr x) (cons (car x) y))))))

Trace the process generated by (foo (list 1 2 3 4)).
What does foo do to a list?

ANSWER: It reverses the list

Can you think of a definition of foo which achieves the same thing, but creates a recursive process?

ANSWER: (note that this is not as efficient as the iterative definition!)

(define (foo ls)
    (if (null? ls)
        nil
        (append (foo (cdr ls)) (list (car ls)))))