**Continuation datatype starting code**

(define read-flatten-print

(lambda ()

(display "enter slist to flatten: ")

(let ([slist (read)])

(unless (eq? slist 'exit)

(flatten-cps slist

(lambda (val)

(pretty-print val)

(read-flatten-print)))))))

(define flatten-cps

(lambda (ls k)

(if (null? ls)

(k ls)

(flatten-cps (cdr ls)

(lambda (v) (if (list? (car ls))

(flatten-cps (car ls)

(lambda (u) (append-cps u v k)))

(k (cons (car ls) v))))))))

(define append-cps

(lambda (l1 l2 k)

(if (null? l1)

(k l2)

(append-cps (cdr l1)

l2

(lambda (appended-cdr)

(k (cons (car l1)

appended-cdr)))))))

**Add apply-k:**

(define read-flatten-print

(lambda ()

(display "enter slist to flatten: ")

(let ([slist (read)])

(unless (eq? slist 'exit)

(flatten-cps slist

(lambda (val)

(pretty-print val)

(read-flatten-print)))))))

**(define apply-k**

**(lambda (k v)**

**(k v)))**

(define flatten-cps

(lambda (ls k)

(if (null? ls)

(**apply-k** k ls)

(flatten-cps (cdr ls)

(lambda (v) (if (list? (car ls))

(flatten-cps (car ls)

(lambda (u) (append-cps u v k)))

(**apply-k** k (cons (car ls) v))))))))

(define append-cps

(lambda (l1 l2 k)

(if (null? l1)

(**apply-k** k l2)

(append-cps (cdr l1)

l2

(lambda (appended-cdr)

(**apply-k k** (cons (car l1)

appended-cdr)))))))

**continuation datatype:**

**(load "chez-init.ss")**

**(define-datatype kontinuation kontinuation?**

**[init-k]**

**[flatten-cdr-k**

**(ls list?)**

**(k kontinuation?)]**

**[flatten-car-k**

**(ls list?)**

**(flattened-cdr list?)**

**(k kontinuation?)]**

**[append-k**

**(first symbol?)**

**(k kontinuation?)]**

**)**

**(define apply-k**

**(lambda (k v)**

**(cases kontinuation k**

**[init-k ()**

**(begin (pretty-print v)**

**(read-flatten-print))]**

**[flatten-cdr-k (ls k)**

**(if (null? ls)**

**(apply-k kontinuation ls)**

**(flatten-cps (cdr ls)**

**(if (list? (car ls))**

**(flatten-cps (car ls)**

**(flatten-car-k ls v k))**

**(apply-k k (cons (car ls) v)))))]**

**[flatten-car-k (ls flattened-cdr k)**

**(append-cps v flattened-cdr k)]**

**[append-k (first k)**

**(apply-k k (cons first v))]**

**)))**

(define read-flatten-print

(lambda ()

(display "enter slist to flatten: ")

(let ([slist (read)])

(unless (eq? slist 'exit)

(flatten-cps slist **(init-k)**)))))

(define flatten-cps

(lambda (ls k)

(if (null? ls)

(apply-k k ls)

(flatten-cps (cdr ls) **(flatten-cdr-k ls k)**))))

(define append-cps

(lambda (l1 l2 k)

(if (null? l1)

(apply-k k l2)

(append-cps (cdr l1)

l2

**(append-k (car l1) k)**))))