CS61A Exam Prep 12

Summer 2016 Final Q6c

(c) **(3 pt)** For each of the following Scheme expressions, circle the correct number of calls that would be made to `scheme_eval` and `scheme_apply` when evaluating the expression in our Scheme interpreter (Project 4). Assume you are **not** using the tail-recursive `scheme_optimized_eval`.

```
(- (* 6 11) 2 2 2)
```

Number of calls to `scheme_eval`:	1	4	7	9	12
Number of calls to `scheme_apply`:	0	1	2	3	4

```
(if (+ 0 (- 1 1)) (+ 4 5) 4)
```

Number of calls to `scheme_eval`:	1	4	7	9	12
Number of calls to `scheme_apply`:	0	1	2	3	4

```
((lambda (x) (* x x)) 57)
```

Number of calls to `scheme_eval`:	1	4	7	9	12
Number of calls to `scheme_apply`:	0	1	2	3	4

Fall 2016 Final Q6a + Q6b (append '(3 4) '(1 2))

(3 4 1 2)

The built-in **append** procedure is equivalent in behavior to the following definition.

```
(define (append s t) (if (null? s) t (cons (car s) (append (cdr s) t))))
```

(a) (1 pt) Circle *True* or *False*: The recursive call to ~~append in the definition above~~ is a tail call.

(b) (4 pt) Implement `atoms`, which takes a Scheme expression. It returns a list of the non-`nil` atoms contained in the expression in the order that they appear. A non-`nil` atom is a number, symbol, or boolean value.

```
scm> (atoms 1)
(1)
scm> (atoms '(+ 2 3))
(+ 2 3)
scm> (atoms '(+ (* 2 3) 4))
(+ * 2 3 4)
scm> (atoms '(* (+ 1 (* 2 3)) (+ 4 5)))
(* + 1 * 2 3 + 4 5)
```

(2 3)

(atom? x)

```
(define (atoms exp)
    (cond ((null? exp) ____nil____ )
          ((atom? exp) ____(list exp) or (cons exp nil) or '(exp)____ )
          (else (append ____(atoms (car exp))____ ____(atoms (cdr exp))____ ))))
```

(+) (* 2 3 4)

Fall 2016 Final Q6c

(c) (5 pt) If Scheme had only numbers and two-argument procedures, parentheses would be unnecessary. To demonstrate, implement `tally`, which takes the list of atoms in a Scheme expression. It returns a list whose first element is the value of the original expression. Assume that the original expression consists only of numbers and call expressions with arithmetic operators (such as `+` and `*`) and exactly two operands.

Hint: `tally` is similar to the built-in `eval` procedure: `(eval '(+ (* 2 3) 4))` evaluates to 10.

```
scm> (car (tally '(1)))                  ; atoms in 1
1
scm> (car (tally '(+ 2 3)))              ; atoms in (+ 2 3)
5
scm> (car (tally '(+ * 2 3 4)))          ; atoms in (+ (* 2 3) 4)
10
scm> (car (tally '(* + 1 * 2 3 + 4 5))) ; atoms in (* (+ 1 (* 2 3)) (+ 4 5))
63
```

```
(define (tally s)
  (if (number? (car s))  ____________________________________________________
      (let ((first  ____________________________________________ ))
        (let ((second  ____________________________________________ ))
          (cons ( ____________ ( ____________ (car s) (car first) (car second)))
                ____________________________________________ )))))
```

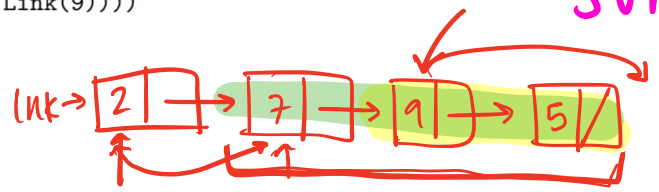
Implement a function **bouncer** that takes in a linked list and an index i and moves the value at index i of the link to the end. You should mutate the input link.

You should implement **swapper** to help with your implementation.

```
def bouncer(link, k):
    """
    >>> lnk = Link(5, Link(2, Link(7, Link(9))))
    >>> bouncer(lnk, 0)
    >>> lnk
    Link(2, Link(7, Link(9, Link(5))))
    >>> bouncer(lnk, 2)
    >>> lnk
    Link(2, Link(7, Link(5, Link(9))))
    """
    if ____k == 0____:
        ____swapper(link)____
    else:
        ____bouncer(link.rest, k-1)____

def swapper(link):
    if ____link is Link.empty or link.rest is Link.empty____:
        return
    ____link.first____, ____link.rest.first____ = ____link.rest.first____, ____link.first____
    ____swapper(link.rest)____
```

k = 2 & 0



Summer 2019 Final Q5

go.cs61a.org/melanie-feedback

melaniecooray@