Some More Practice Questions

Spring 2020

This is a selection of questions that might be similar, in broad terms, to exam questions.

There is no expectation that all topics covered here are on the exam.

There is no expectation that all topics on the exam are covered here.

LLT

Exercise

Write a function `sum-even-llt` that consumes a `LLT` and returns the sum of all the even leaves.
For example, `(sum-even-ll (list 2 (list 3 5) (list 91 (list 4 6)))) => 12`

```
(define (corge L n)
  (cond [(empty? L) '()]
        [(even? (first L))
         (cons (first L)
               (corge (rest L) (+ 1 (remainder n 2))))]
        [else
         (cons (+ n (first L))
               (corge (rest L) (+ 1 (remainder n 2))))]))

(corge (list 3 3 2 2) 1)
```

Exercise

By hand, determine what is `(corge (list 3 3 2 2) 1)` returns.
(You may verify your answer by running the code.)

```
(define-struct node (key val left right))

;; A binary search tree (BST) is either
;; * '() or
;; * (make-node Nat Any BST BST)...
;; which satisfies the ordering property recursively:
;; * every key in left is less than key
;; * every key in right is greater than key
```

Exercise

Write a function `(count-smaller B value)` that consumes a `BST` and returns the number of nodes where the key is less than `value`. For example,

```
(define dict1
  (make-node 10 "ten"
             (make-node 5 "five" '() '())
             (make-node 15 "fifteen" '() '())))

(check-expect (count-smaller dict1 16) 3)
(check-expect (count-smaller dict1 14) 2)
(check-expect (count-smaller dict1 9) 1)
(check-expect (count-smaller dict1 4) 0)
```

! Only recurse on subtrees that could potentially have one or more such node.

Exercise

Determine what `(fun-a 6)` returns.

```
(define (fun-a n)
  (local [(define a (range 0 n 1))
          (define b (filter even? a))
          ]
    (foldr + 7 b)))
```

```
(define-struct binode (op arg1 arg2))
;; a binary arithmetic expression internal node (BINode)
;; is a (make-binode Operator BinExp BinExp)

;; A binary arithmetic expression (BinExp) is either:
;;   a Num or
;;   a BINode
```

Exercise

1. Draw a picture of the tree corresponding to the expression $((2 \times 6) + (5 \times 2)) \div (5 - 3)$
2. Write the Racket expression corresponding to that tree.

Exercise

1. Determine the mathematical expression corresponding to the Racket expression:

```
(make-binode '*
             (make-binode '- 7 5)
             (make-binode '+
                          (make-binode '* 2 5)
                          (make-binode '+ 3 4)))
```

2. Draw a picture of the tree corresponding to the expression.

Ex. Complete `collection-price` by writing its body.

```
(define-struct book (title author price))
;; a Book is a (make-book Str Str Num)
;; Requires: price >= 0

;; (collection-price author catalog) return the cost of
;;   buying all the books in catalog written by author.
;; collection-price: Str (listof Book) -> Num
;; Example:
(define library (list (make-book "Green Eggs and Ham" "Seuss" 11.69)
                      (make-book "Red Planet" "Heinlein" 19.31)
                      (make-book "Fox in Socks" "Seuss" 11.18)
                      (make-book "Democracy and Education" "Dewey" 8.81)
                      (make-book "Starman Jones" "Heinlein" 9.99)))
(check-expect (collection-price "Seuss" library) 22.87)
(check-expect (collection-price "Heinlein" library) 29.30)
(check-expect (collection-price "King" library) 0)
```

In this question you will write a function that consumes a `(listof Num)` and returns a new list, where the distance from the end has been added to each value.

For example, `(add-distance-to-end (list 2 3 5 7 11)) => (list 6 6 7 8 11)`

Since the distance from 11 to the end is zero, so it is unchanged; the distance from 7 to the end is 1, so it becomes 8; the distance from 5 to the end is 2, so it becomes 7, etc.

Ex. Write `add-distance-to-end` without using recursion, using higher order functions such as `map`, `foldr`, and `filter`.

Ex. Write `add-distance-to-end2` using recursion, without using any higher order functions.

Determine the fully simplified value of each expression.

Exercise

```
(define (func-a L)
  (local
    [(define (f x) (+ x (first L)))]
    (map f (range 0 (length L) 1))))

(func-a (list 2 3 5 7 11))
```

Exercise

```
(define (func-b L M)
  (local
    [(define (h a b)
       (cond [(even? a) (cons a b)]
             [else
              (cons (* 2 a) b)]))]
    (foldr h M L)))

(func-b (list 1 2 3 4) (list 1 2 3 4))
```

Determine the fully simplified value of each expression.

Exercise

```
(define (func-c L)
  (foldr (lambda (a b)
           (cons (range 0 a 1) b))
         '() L))

(func-c (list 2 3 0))
```

Exercise

```
(define (func-d L)
  (local [
          (define (q n)
            (cond [(= n 0) 1]
                  [else
                   (* n (q (- n 1)))]))]
    (map q L)))

(func-d (list 3 4 1))
```

Exercise

Write a function `(pyramid lo hi)`. It consumes two `Nat`, and returns a list contaning the values counting up from `lo` to `hi`, then back down to `lo`.
You may assume `lo` is not greater than `hi`.
For example, `(pyramid 2 5) => (list 2 3 4 5 4 3 2)` `(pyramid 7 7) => (list 7)`

! Do not use **range**.

```
(define-struct blnode (left right))
;; a BLNode is a (make-blnode BTL BTL)
;; a Binary Leaf-labelled Tree (BLT) is either
;;   Num or
;;   a BLNode

(define T1 (make-blnode 4 (make-blnode 2 2)))
(define T2 (make-blnode (make-blnode 8 0) T1))
```

Exercise

Write a function `(blt-total T)` that returns the total of all the leaves in `T`. `(blt-total T1) => 8`
`(blt-total T2) => 16`

Less well known than the Fibonacci numbers are the Tribonacci numbers, defined as follows:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \text{ or } n = 2 \\ T(n-1) + T(n-2) + T(n-3) & \text{otherwise.} \end{cases}$$

Ex. Write a function to compute the nth Tribonacci number

Ex. What is $T(5)$?