FINAL REVIEW

CS Scholars

February 25 and February 27, 2017

Instructions: Your team is in charge of one of the following sections: Eval/Apply, Box and Pointer, Environment Diagrams, Streams, Linked Lists, or Trees. You have 1 hour to solve the question(s) in your section. In the last hour, you will present your solution. Think about how you approached the question(s) (write down ideas!) and any other helpful strategies you used to solve it.

Please take a look at all of the exercises after finishing your section. They are independent of each other so order doesn't matter. **These problems are tough!**

1 Eval/Apply

1. How many calls to `scheme-eval`? How many to `scheme-apply`?
   1. `(and 0 0 0 0 #f)`
   2. `(cond (#f (/ 1 1)) ((+ 0) mushrooms) (else (- 0 1))`
   3. `(define (another-castle mario) (if (= mario 0) 0 (another-castle (- mario 1))))`
   4. `(let ((luigi another-castle) (bowser #f)) (or bowser (luigi 1)))`
   5. `What if we replace bowsers binding with #t?`

2 Box and Pointer

2. Lets cons-struct some Scheme lists! How would you represent the following lists with box-and-pointer diagrams? Draw a diagram for each.

   1. `'(1 2 (3) 4 (5 . 6))`

   2. `'(cons 1 (cons 2 (cons (1 . 2) nil)))`

3. Assume `cons-all` takes a list of lists and inserts a value at the front of every list. Draw the box and pointer that results from executing the code below.

```
(define (partitions n)
   (define (part i lst)
      (if (= i 0) lst
      (part (- i 1) (cons (list i) (cons-all i lst)))))
   (part n nil))
(partitions 5)
```

3.1 Questions

4. Draw an environment diagram for the following code.

```
b, c = 4, 5
def a(b):
    start = a
    def c():
        nonlocal c, start
        if start != 1:
            start = 1
            c, c = 1, c()
        return b(2)
    c()
    return lambda y: c.append(1)

a(lambda x: [b + x])(3)
```

4.1 Questions

5. Suppose we have two streams S and T, where the elements of each sequence are represented as follows:

$$S = S_0, S_1, S_2, \ldots$$

$$T = T_0, T_1, T_2, \ldots$$

Now imagine each elements of the above stream paired up as shown in the infinite matrix below:

(S_0, T_0)	(S_0, T_1)	(S_0, T_2)	...
(S_1, T_0)	(S_1, T_1)	(S_1, T_2)	...
(S_2, T_0)	(S_2, T_1)	(S_2, T_2)	...
$\vdots$	$\vdots$	$\vdots$	$\ddots$

We wish to generate a stream that contains all the pairs in the array that lie on or above the diagonal, i.e. the pairs:

(S_0, T_0)	(S_0, T_1)	(S_0, T_2)	...
	(S_1, T_1)	(S_1, T_2)	...
		(S_2, T_2)	...
			$\ddots$

Now fill in the blanks to `pairs`, which takes two streams and outputs a stream of all possible pairs of elements from these two streams above and including the diagonal. For example, pairs(positives, positives) should return a stream starting with:

$$(1,1), (1,2), (2,2), (1,3), (2,3), \ldots$$

```
def pairs(s, t):

    top = stream_map(____________________, ____________________)

    rest = lambda: interleave(____________________, ____________________)

    return Stream(____________________, ____________________)
```

Here is the `Link` class, provided for your reference.

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

6. Reverse a shallow linked list.

```
def reverse(LL):

    last = _____________________________

    current = _____________________________

    while (_____________________________):

        nxt = _____________________________

        current.rest = _____________________________

        last = _____________________________

        _____________________________
    return last
```

What is the runtime of your solution?

Here is the `Link` class, provided for your reference.

```
class Tree(object):
    """ A tree with internal values. """

    def __init__(self, entry, left=None, right=None):
        self.entry = entry
        self.left = left
        self.right = right
```

7. In this problem, the input is a binary tree and the output is a list. The list is formed by getting the rightmost entry first.

```
def tree_to_reversed_list(tree):
    """
    >>> t = Tree(5, Tree(1, None, Tree(4)), Tree(7, Tree(6),
       Tree(8)))
    >>> tree_to_reversed_list(t)
    [8, 7, 6, 5, 4, 1]
    """
    lst = ____________________________

    if ____________________________________:

        if ____________________________________:

            lst.extend(____________________________)

        ____________________________________

        if ____________________________________:

            lst.extend(____________________________)

    return lst
```