FINAL REVIEW 12

COMPUTER SCIENCE 61A

April 30, 2015

1 Dynamic Scoping

```
scm> (define f (mu (x) (mu (y) (- x y))))
scm> (define g (lambda (x) (lambda (y) ((f y) x))))
scm> (define h (mu (x) (lambda (y) (g ((f x) y)))))
scm> ((f 6) 1)

scm> ((g 6) 1)

scm> ((h 6) 1)

scm> (((h 6) 1) 1)
```

Solution:

```
Unknown identifier 'x'
0
(lambda (y) ((f y) x))
0
```

2 Iterables and Generators

1. Modify the `Link` class so that it supports the iterable interface. Hint: use the `yield` statement to make the `__iter__` method a generator function.

   ```
   class Link:
   ```

```
    empty = ()
    def __init__(self, first, rest=empty):
        self.first, self.rest = first, rest
```

Solution:

```
    def __iter__(self):
        curr = self
        while curr != Link.empty:
            yield curr.first
            curr = curr.rest
```

3 Streams

1. Implement the `tree_to_stream` function, which takes in a Tree `t` and returns a Stream that gives back the elements of `t` one by one. The Stream should be ordered so that parent values come before child values. You may find the provided `append_streams` function helpful. See the example for more details.

```
>>> t = Tree(1, [Tree(3, [Tree(4), Tree(5)]), Tree(2)])
>>> s = tree_to_stream(t)
>>> print_stream(s)
< 1 3 4 5 2 >
```

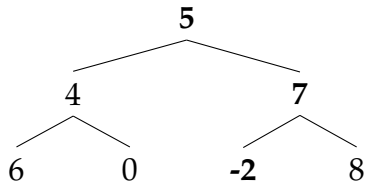
```
def append_streams(s1, s2):
    """
    Returns a new stream of s1's elements, then s2's.
    >>> s1 = Stream(1, lambda: Stream(2))
    >>> s2 = append_streams(s1, s1)
    >>> print_stream(s)
    < 1 2 1 2 >
    """
    if s1 is Stream.empty:
        return s2
    return Stream(s1.first,
                  lambda: append_streams(s1.rest, s2))

def tree_to_stream(t):
```

Solution:

```
def compute_rest():
    rest = Stream.empty
    for b in t.branches:
        rest = append_streams(rest, tree_to_stream(b))
    return rest
return Stream(t.entry, compute_rest)
```

4 Scheme

1. Write a Scheme function `follow-path-sum` that takes a binary tree `t` and a list `l` that consists of Boolean values, i.e. `#t` and `#f`, and returns the sum along the path of `t` specified by `l`, where `#t` specifies left and `#f` specifies right. After you write this function, try to reimplement it tail recursively. For example,




```
scm> (follow-path-sum my-tree '(#f #t))
10
```

For your reference:

```
(define (make-btree entry left right)
  (cons entry (cons left right)))
(define (entry tree) (car tree))
(define (left tree) (car (cdr tree)))
(define (right tree) (cdr (cdr tree)))

(define my-tree
  (make-btree 5
    (make-btree 4 (make-btree 6 nil nil) nil)
    (make-btree 7 (make-btree -2 nil nil) (make-btree 8 nil
      nil))))
```

Solution:

```
(define (follow-path-sum t l)
  (if (null? l) (entry t)
      (let ((next-tree (if (car l) (left t) (right t))))
        (+ (entry t) (follow-path-sum next-tree (cdr
          l))))))
```

Tail recursive solution:

```
(define (follow-path-sum t l)
  (define (follow-path-sum-iter t l so-far)
    (if (null? l) (+ so-far (entry t))
```

```
    (let ((next-tree (if (car l) (left t) (right t))))
      (follow-path-sum-iter next-tree (cdr l) (+ (entry
        t) so-far)))))
(follow-path-sum-iter t l 0))
```

5 SQL

1. Suppose we are trying to calculate the degrees of separation between two users on Facebook. For example, if Clinton and Barack are friends, and Barack and Abraham are friends, then Abraham is 2 degrees of separation away from Clinton. We are given a table `friends` with the columns `u1` and `u2`. Create a new table `deg_sep` with the columns `u1`, `u2`, and `n`, where `n` is the degree of separation between `u1` and `u2`. Limit the number of rows so that the maximum degree of separation is 10. Notice that Abraham could also be considered 2 degrees of separation away from himself.

friends	
u1	u2
Abraham	Barack
Barack	Abraham
Barack	Clinton
Clinton	Barack

deg_sep		
u1	u2	n
Abraham	Barack	1
Barack	Abraham	1
Barack	Clinton	1
Clinton	Barack	1
Abraham	Abraham	2
Abraham	Clinton	2
...		

Solution:

```
CREATE TABLE deg_sep AS
    WITH deg_sep(u1, u2, n) AS (
        SELECT u1, u2, 1 FROM friends UNION
        SELECT deg_sep.u1, friends.u2, n+1 FROM friends,
           deg_sep
            WHERE deg_sep.u2 = friends.u1 AND n < 10
    )
    SELECT * FROM deg_sep;
```

2. In our previous question, the resulting table included all possible degrees of separation for two users. Now create a table `min_deg_sep` that includes only the minimum degree of separation for 2 users. You may use `deg_sep` from above.

Solution:

```
CREATE TABLE min_deg_sep AS
    SELECT u1, u2, MIN(n) FROM deg_sep GROUP BY u1, u2;
```