Mutable Linked Lists: exam-level questions

> If you need help reviewing Mutable Linked Lists, take a look at these resources:
>
> - Albert's and Robert's slides (https://docs.google.com/presentation/d/18KropyOg2nZf8e8Ot0JAqxTh0mQp5WnfjeU4dc5cUkU/edit)

For this section, we will be using the `Link` class implementation from lecture:

```
class Link(object):
    empty = ()

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

    def__len__(self):
        return 1 + len(self.rest)

    def __getitem__(self, i):
        if i == 0:
            return self.first
        return self.rest[i - 1]

    def __repr__(self):
        if self.rest is empty:
            return 'Link({})'.format(repr(self.first))
        return 'Link({}, {})'.format(repr(self.first),
                                     repr(self.rest))
```

Each question has a "Toggle Solution" button -- click it to reveal that question's solution.

Code-Writing questions

Question 1

Implement a function `validate`, which takes a Link and returns True if the Link is valid.

```
def validate(lst):
    """Returns True if lst is a valid Link.

    >>> lst = Link(1, Link(2, Link(3)))
    >>> validate(lst)
    True
    >>> okay = Link(Link(1), Link(2))
    >>> validate(okay)
    True
    >>> bad = Link(1, 2)
    >>> validate(Link.empty)
    True
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 2

Implement a function `count`, which takes a Link and another value, and counts the number of times that `value` is found in the Link.

```
def count(r, value):
    """Counts the number of times VALUE shows up in R.

    >>> r = Link(3, Link(3, Link(2, Link(3))))
    >>> count(r, 3)
    3
    >>> count(r, 2)
    1
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 3

Implement a function `extend_link`, which takes two Links, `s1` and `s2`, and mutates `s1` such that it contains the elements of `s2` at its tail. Do this mutatively — don't return anything! Also, make sure `s2` itself does not get attached to `s1`. You may assume `s1` always has at least one element.

```
def extend_link(s1, s2):
    """Extends s1 to include the elements of s2.

    >>> s1 = Link(1)
    >>> s2 = Link(2, Link(3))
    >>> extend_link(s1, s2)
    >>> s1
    Link(1, Link(2, Link(3)))
    >>> s1.rest is not s2
    True
    """
    "*** YOUR CODE HERE ***"
```

Hint: This question is similar to the `extend_link` from lecture, except this version mutates the original Link and does not make `s2` part of `s1`."""",

Toggle Solution

Question 4

Implement a function `deep_map`, which takes an (possibly nested) Link and a function `fn`, and applies `fn` to every element in the Link. If an element is itself a Link, recursively apply `fn` to each of the element's elements.

```
def deep_map(fn, lst):
    """Applies FN to every element in lst.

    >>> normal = Link(1, Link(2, Link(3)))
    >>> deep_map(lambda x: x*x, normal)
    >>> normal
    Link(1, Link(4, Link(9)))
    >>> nested = Link(Link(1, Link(2)), Link(3, Link(4)))
    >>> deep_map(lambda x: x*x, nested)
    >>> nested
    Link(Link(1, Link(4)), Link(9, Link(16)))
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution