FINAL REVIEW PART 1 (PRE-MT2) Solutions

COMPUTER SCIENCE MENTORS

April 19, 2021 - April 21, 2021

1 Environment Diagrams

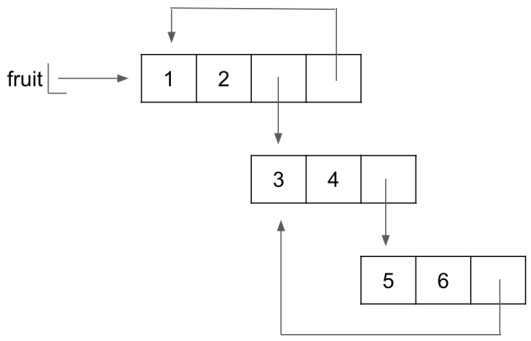
1. Draw the environment diagram that results from running the following code.

```
def f(f):
    def h(x, y):
        z = 4
        return lambda z:  (x + y) * z

    def g(y):
        nonlocal g, h
        g = lambda y: y[:4]
        h = lambda x, y: lambda f: f(x + y)
        return y[3] + y[5:8]

    return h(g("sarcasm!"), g("why?"))

f = f("61a")(2)
```

https://tinyurl.com/y56ezjz9

2. Fill in each blank in the code example below so that its environment diagram is the following. You do not need to use all the blanks.

```
fruit = [1, 2, [3, 4]]
fruit._____________________________
fruit[3][2]._____________________
fruit[2][2]._____________________
fruit[3][3][2][2][2][1] = ____
```




```
fruit = [1, 2, [3, 4]]
fruit.append(fruit)
fruit[3][2].append([5, 6])
fruit[2][2].append(fruit[2])
fruit[3][3][2][2][2][1] = 4
```

2 OOP

1. The `DLList` class is a spin off of the normal `Link` class we learned in class; each `DLList` link has a `prev` attribute that keeps track of the previous link and a **`next`** attribute that keeps track of the next link. Fill in the following methods for the `DLList` class.

   (a)
```
class DLList:
    """
    >>> lst = DLList(6, DLList(1))
    >>> lst.value
    6
    >>> lst.next.value
    1
    >>> lst.prev.value
    AttributeError: 'NoneType' object has no attribute 'value
       '
    """
    empty = None
    def __init__(self, value, next=empty, prev=empty):

        _____

        _____

        _____
```

```
    def __init__(self, value, next=empty, prev=empty):
        self.value = value
        self.next = next
        self.prev = prev
```

   (b)
```
    def add_last(self, value):
        """
        >>> lst = DLList(6)
        >>> lst.add_last(1)
        >>> lst.value
        6
        >>> lst.next.value
        1
        >>> lst.next.prev.value
        6
        """
```

```
        pointer = self
        while _____________________________________:

          ___________________________________________

        _________________ = DLList(_____________________________________)
```

```
    def add_last(self, value):
      pointer = self
      while pointer.next != DLList.empty:
        pointer = pointer.next
      pointer.next = DLList(value, DLList.empty, pointer)
```

(c)

```
    def add_first(self, value):
      """
      >>> lst = DLList('A')
      >>> lst.add_first(1)
      >>> lst.value
      1
      >>> lst.next.value
      'A'
      >>> lst.next.prev.value
      1
      >>> lst.add_first(6)
      >>> lst.value
      6
      >>> lst.next.next.prev.value
      1
      """
      old_first = DLList(_____________________________)

      _________________ = _____________________________________

      _________________ = _____________________________________

      if ___________________________________:

        ___________________________________________________________
```

```
    def add_first(self, value):
      old_first = DLList(self.value, self.next, self)
      self.value = value
      self.next = old_first
```

```
        if old_first.next != DLList.empty:
          old_first.next.prev = old_first
```

CSM 61A Spring 2021: Ivan Penev and Joshua Baum, with
Jamie Ip, Kenneth Chi, Kevin Moy, Mathilde McKeever, Cyrus Bugwadia, Peter Zhang, and
Kunal Agarwal, Luke Liu, John So, Jemmy Zhou, Ruslana Yurtyn, Vivian Lu, Matt Au, Lily Yang, Mihira Patel, Jessica Yu, Aidan Tong, Matthew Guo, Nikhita Anumukonda, Laryn Qi, Grace Yi, Jennifer Huang, Cindy Lin, James Fletcher, Sean Sananikone, Abinaya Srikant, Uma Unni

3 Complexity

1. **Fast Exponentiation:** in this problem, we will examine a real-world algorithm used to improve the speed of calculating exponents.

   (a) First, express the runtime of the naive exponentiation algorithm in big-θ notation.

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n - 1)
```

   $\theta(n)$. n decreases by 1 each call, so there are naturally n calls.

   (b) Now, express the runtime of the fast exponentiation algorithm in big-θ notation.

```
def fast_exp(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0: # Assume square runs in constant time
        return square(fast_exp(b, n // 2))
    else:
        return b * fast_exp(b, n - 1)
```

   $\theta(\log n)$. n is halved each call, so the number of calls is the number of times n must be halved to get to 1. This is $\log n$.

   (c) What about this slightly modified version of `fast_exp`?

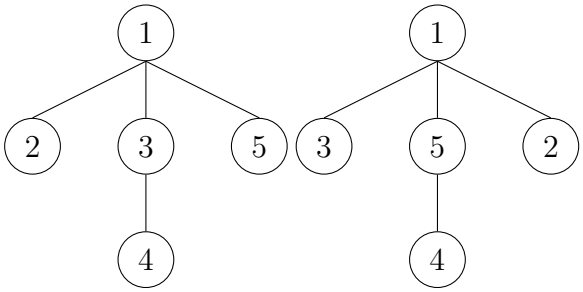
```
def fast_exp(b, n):
    for _ in range(50 * n):
        print("Killing time")
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(fast_exp(b, n // 2))
    else:
        return b * fast_exp(b, n - 1)
```

   $\theta(n)$. Ignore the constant term. The first call will perform n operations, the second call will perform $n/2$ operations, the third will perform $n/4$ operations, etc. Using geometric series, we see this adds up to $2n$, which is n if we ignore constant terms.

4 Trees

1. Implement `rotate`, which takes in a tree and rotates the labels at each level of the tree by one to the left destructively. This rotation should be modular (That is, the leftmost label at a level will become the rightmost label after running rotate). You do NOT need to rotate across different branches.

   For example, given tree `t` on the left, `rotate(t)` should mutate `t` to give us the right.




```
def rotate(t):
    """
    >>> t1 = Tree(1, [Tree(2), Tree(3, [Tree(4)]), Tree(5)])
    >>> rotate(t1)
    >>> t1
    Tree(1, [Tree(3), Tree(5, [Tree(4)]), Tree(2)])
    >>> t2 = Tree(1, [Tree(2, [Tree(3), Tree(4)]),
                      Tree(5, [Tree(6)])])
    >>> rotate(t2)
    >>> t2
    Tree(1, [Tree(5, [Tree(4), Tree(3)]),
                      Tree(2, [Tree(6)])])
    """
    branch_labels = ____________________________________________

    n = len(t.branches)

    for ____________________________________________:

        ____________________________________________________

        ____________________________________________________

        ____________________________________________________
```

```
def rotate(t):
    branch_labels = [b.label for b in t.branches]
    n = len(t.branches)
    for i in range(n):
        branch = t.branches[i]
        branch.label = branch_labels[(i + 1) % n]
        rotate(branch)
```

5 Generators

1. (a) Implement `n_apply`, which takes in 3 inputs `f`, `n`, `x`, and outputs the result of applying `f`, a function, `n` times to `x`. For example, for `n = 3`, output the result of `f(f(f(x)))`.

```
def n_apply(f, n, x):
    """
    >>> n_apply(lambda x: x + 1, 3, 2)
    5
    """

    for _____________________________:

        x = _____________________________

    return _____________________________
```

```
def n_apply(f, n, x):
    for i in range(n):
        x = f(x)
    return x
```

   (b) Now implement `list_gen`, which takes in some list of integers `lst` and a `function f`. For the element at index `i` of `lst`, `list_gen` should apply `f` to the element `i` times and yield this value `lst[i]` times. You may use `n_apply` from the previous part.

```
def list_gen(lst, f):
    """
    >>> a = list_gen([1, 2, 3], lambda x: x + 1)
    >>> list(a)
    [1, 3, 3, 5, 5, 5]
    """
```

```
    for _____________________________:

        yield from [_____________________________________________]

def list_gen(lst, f):
    for i in range(len(lst)):
        yield from [n_apply(f, i, lst[i]) for j in range(lst[i
            ])]
```

2. Complete the implementation of `iter_link`, which takes in a linked list and returns a generator which will iterate over the values of the linked list in order. Your function should support deep linked lists.

```
def iter_link(lnk):
    """
    Yield the values of a linked list in order; your function
       should support deep linked lists.
    >>> lst1 = Link(1, Link(2, Link(3, Link(4))))
    >>> list(iter_link(lst1))
    [1, 2, 3, 4]
    >>> lst2 = Link(1, Link(Link(2, Link(3)), Link(4, Link(5))))
    >>> print(lst2)
    <1 <2 3> 4 5>
    >>> iter_lst2 = iter_link(lst2)
    >>> next(iter_lst2)
    1
    >>> next(iter_lst2)
    2
    >>> next(iter_lst2)
    3
    >>> next(iter_lst2)
    4
    """
    if lnk is not Link.empty:

        if type(_____________________) is Link:

            ____________________________________________

        else:

            ____________________________________________

        ____________________________________________
```

```
def iter_link(lnk):
    if lnk is not Link.empty:
        if type(lnk.first) is Link:
            yield from iter_link(lnk.first)
        else:
            yield lnk.first
        yield from iter_link(lnk.rest)
```

CSM 61A Spring 2021: Ivan Penev and Joshua Baum, with
Jamie Ip, Kenneth Chi, Kevin Moy, Mathilde McKeever, Cyrus Bugwadia, Peter Zhang, and
Kunal Agarwal, Luke Liu, John So, Jemmy Zhou, Ruslana Yurtyn, Vivian Lu, Matt Au, Lily Yang, Mihira Patel, Jessica Yu, Aidan Tong, Matthew Guo, Nikhita Anumukonda, Laryn Qi, Grace Yi, Jennifer Huang, Cindy Lin, James Fletcher, Sean Sananikone, Abinaya Srikant, Uma Unni