CS 120: Intro to Computer Programming II

In-Class Activity - 03 Linked Lists - Day 2

Activity 1 - Turn in this one

How do we represent an empty linked list?

A common misconception: many students think that an "empty list" would be a single node, but that the node would have a value of `None`. Discuss with your group why this isn't actually an empty linked list. What really is this??? Could you come up with an array, which represents the same values as this linked list?

Activity 2 - Turn in this one

I've discussed a couple of different valid ways to represent `None` in a reference diagram. What are they? Draw an example of each.

Activity 3 - Turn in this one

We're starting to talk about how to "traverse" a list - that is, how to iterate through it. I said that a common mistake was to write the following line of code:

```
head = head.next
```

(Sometimes, this is exactly what you want - but often, it's a easy mistake that students make.)

This activity has several parts:

- First, draw a linked list with exactly 3 nodes. Make sure to include the head pointer.
- Next, execute the following code. Draw a new reference diagram after **every line**:

```
head = head.next
head = head.next
head = head.next
```

- Discuss with your group: what is happening at each line of code? What happens to the old head, and the other nodes, as the head pointer moves?

(activity continues on the next page)

Activity 4 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

For this activity, draw a picture of a linked list, this time with 5 nodes. Give a value in each node, and include the head pointer. Turn in this picture as part of your solution to this ICA (but you'll be happy to know that I'm not going to make you redraw it - one copy is enough).

Execute the following code on your list. What does it print out?

```
cur = head
print(cur.val)

cur = cur.next
print(cur.val)

cur = cur.next
print(cur.val)

cur = cur.next
print(cur.val)

cur = cur.next
print(cur.val)

cur = cur.next
```

What is `cur` pointing to, at the end of this block of code?

Activity 5 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

When you have a `cur` pointer pointing into a list, I've said that it is **impossible** (using `cur`) to access anything that's **earlier** in the list. Why is this?

Imagine that you somehow made a mistake, and moved `cur` too far down the list. Is there any way to still access the nodes that come before `cur`, or are they lost forever? (Hint: they aren't!) How could you go back to the old nodes, that you've bypassed?

Challenge Activity - Do not turn in this one

Suppose you've been given a linked list, and you want to print out the first three values in it, but you don't know how long the list is. Maybe it's actually shorter than that, or even empty!

Write a snippet of code which will print the first three values of the list (or however many exist, if the list is shorter than that). But do it **without using a loop.** Can you do it simply with `if` statements? How simple, and clear, can you make the code?