CS 120: Intro to Computer Programming II

In-Class Activity - 03 Linked Lists - Day 4

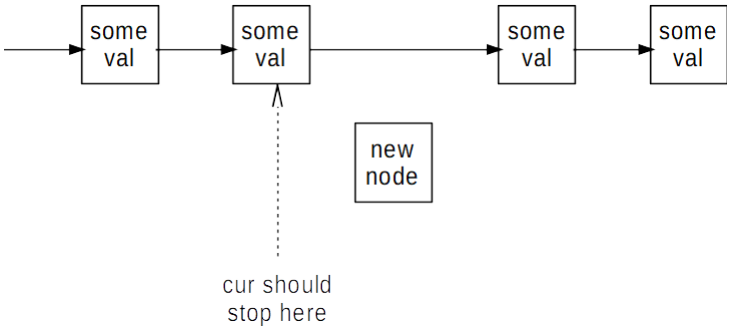
In the video before this lecture, you saw my solution for how you would insert a new node into a **sorted** linked list. It searches the list, and when it finds the correct location, it updates the various `next` pointers.

In this ICA, you are going to re-create this function from scratch. Resist the temptation to just look up the "right" solution - I want you to really dig into this problem, and understand it, so that you can recreate (or modify it) as necessary, throughout the semester.

Activity 1 - Turn in this one

For the first activity, let's focus on the `while` loop, which we'll use to search for the proper place. For now, let's assume that the list is **not** empty, and also that the new node does **not** belong at the very head of the list.

Your goal is to advance `cur` through the list, until it is pointed at the node right **before** where the new node should be inserted, like this:



Discuss with your group what the `while` loop condition should be. That is, what needs to be true, over and over, as `cur` advances - and how will you know when to stop?

Turn in (a) the description of what is always true about nodes as you move past them (until you stop); and (b) the code of the `while` loop.

Solution: We want to keep moving forward, so long as the values (stored in the nodes in the list) are less than the value of the new node we're inserting. Since the list is sorted, we know that **all** of the "less than" nodes will be at the beginning of the list, and all of the "greater than" nodes will be at the end.

We want to stop advancing `cur` when `cur` is still **LESS** than the value we are inserting, but the next node is **MORE.**

```
while cur.next.val < new_node.val:
    cur = cur.next
```

Activity 2 - Turn in this one

Once you have advanced `cur` to the proper location (and stopped there), it only takes two lines of code to actually insert the new node into the list. Give these two lines of code.

Then, discuss with your group - what happens if you get these lines out of order? The answer you turn in should include both code, and also this explanation.

Solution:

```
new_node.next = cur.next
cur.next = new_node
```

If we get these backwards, then we will lose the entire rest of the list! That is, if we change `cur.next`, then we will lose that node - and everything that follows it. But, in the **correct** arrangement, we have already saved that reference (linking to it from `new_node`), **BEFORE** we change `cur`.

Please save your code for next time. You will be extending it during the next ICA.