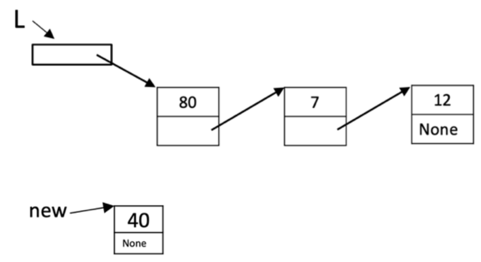
CSC 120 **ICA-14**

Work with your neighbor. (This will be graded for participation only.)

1. Here is a *first pass* at a method to add a new node to the end of a `LinkedList`:

```
def add_to_end(self, new):
    current = self._head
    prev = None             # initialize prev
    while current != None:
        prev = current      # keep track of previous node
        current = current._next
    prev._next = new        # add new to the end
```

Suppose we have the linked list and new node shown below:



Walk through the code for the call `L.add_to_end(new)`. (Remember that `self` will refer to `L`.) Draw the diagrams showing the `current` and `prev` references.

2. Download the code for ICA-14 from the class website: **ICA-14-starter.py**. This code has the classes defined for `LinkedList` and `Node`.

   a) In `main()`, create a linked list called `my_ll.` Create a node that has an integer as a value. Add that node to `my_ll`. Do this two more times so that `my_ll` has three elements that are all **integers**.
   b) Use the method `print_elements()` to print out the linked list elements.

      Note: This method prints the `_value` attribute of each node on a separate line.

   c) Next, print the linked list `my_ll` using this line of code:

      ```
      print(my_ll)
      ```

      Note: We know that `print()` will use the `__str__()` method defined in the class. Take a close look at `__str()__` in the `LinkedList` class. Notice that it loops through the linked list and calls `str()` on each node.

   d) Take a pic of the **output** generated for this problem so far to use as the solution to this problem. (If you don't have your laptop, write out what the code for main would be.)

3. Define a new method called `incr(self)` that increments each element of a linked list by 1. Use `print_elements()` as a guide for how to iterate through a linked list.

   a) Call `incr()` on your linked list.

b) Use `print()` to show how the linked list elements have been modified.
c) Take a pic of **output** generated for this problem so far to use as the solution to this problem. (Or write the code for `incr()` below.)

4. Define a new method called `replace(self, val1, val2)` that iterates through a linked list and replaces all of the `_value` attributes that equal `val1` with `val2`.

   a) Call `replace()` on your linked list.
   b) Use `print()` to show how the linked list elements have been modified.

5. Type in the code for `add_to_end(self, new)`. See slide 106 for reference.

a) Create a new node `n` and call `add_to_end(n)` to add that to your linked list.
b) Use `print()` to show how the linked list has changed.

6. **Challenge.** Write a method `remove_first(self)` that removes the first element of a linked list and returns the node removed. If the list is empty, the method returns `None`.

   a) Call `remove_first()` on your linked list.
   b) Use `print()` to show how the linked list has changed.