CSC 120 **ICA-15**

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

1. Below is the code from last time to add to the end of a linked list:

```
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
```

Modify the code `add_to_end(self, new)` to handle the case of inserting into an empty list. **Hint:** If the list is empty, the code would be just like adding to the front of a linked list.
ANS:

```
def add_to_end(self, new):
    if self._head == None:       # the list is empty
        self._head = new         # add new to the front
    else:
        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
```

2. Below is the code for `get_element(self, n)`, which returns a reference to the node at position `n` in a linked list:

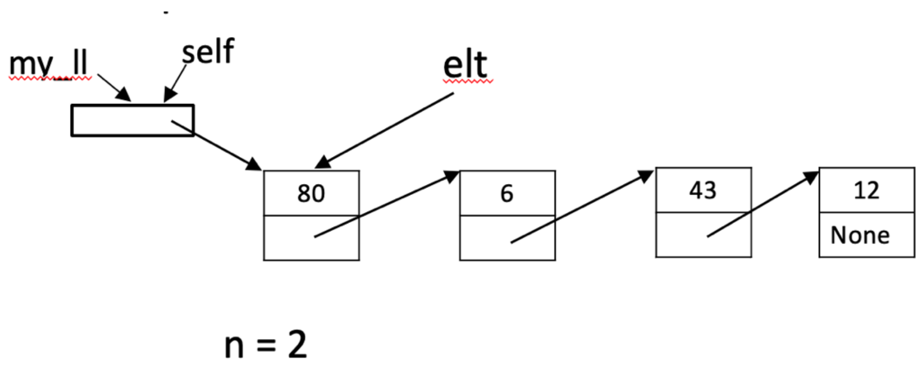
```
def  get_element(self, n):
     elt = self._head
      while elt != None and n > 0:
          elt = elt._next
          n -= 1

     return elt
```

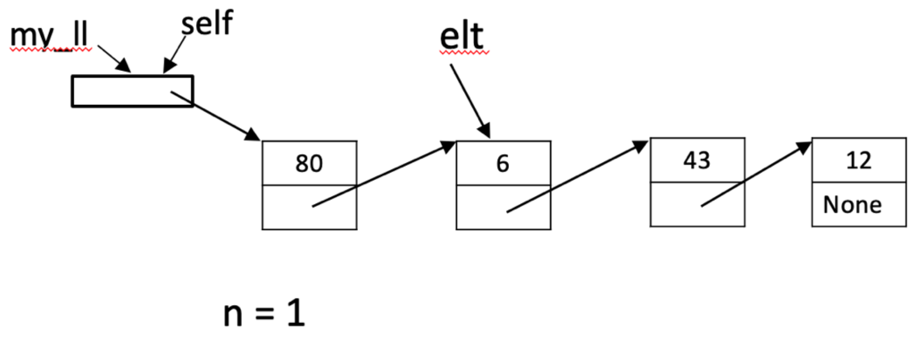
Draw a diagram for a linked list called `my_ll` that has four elements. Walk through the code for `my_ll.get_element(2)` and show on the diagram how the variable `elt` advances; show what that value of `n` is when the element at position 2 is reached and show which node is referred to by `elt` (which is the node that is returned).

ANS:

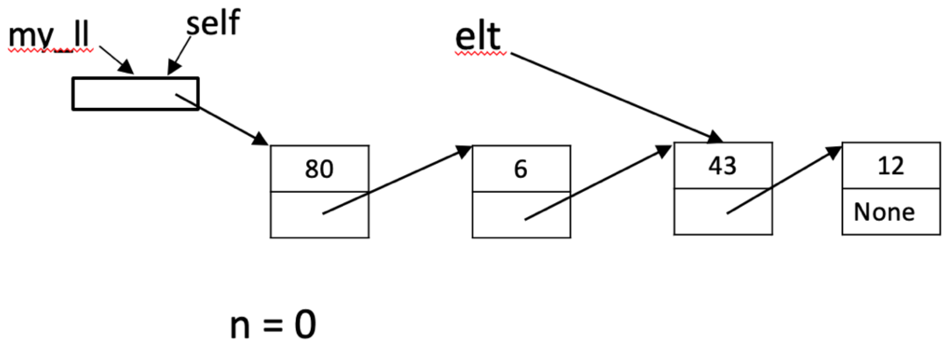
After the first line of code is executed, the diagram looks like this:



After the loop is executed once, the diagram looks like this:



After the loop is executed twice, the diagram looks like this:



After this iteration, the condition of the while loop fails since n == 0 and `elt` is returned by the method. Note that `elt` is a reference to the 3rd element of the list.

Are there any issues with this code? What happens if you ask for the node at position 6 in `my_ll`?

ANS:

There are no issues, per se. But it doesn't exactly match the behavior of indexing for a Python list. In Python, referencing beyond the length of the list will result in a runtime error. This function with return `None` for the call `my_ll.get_element(6)`.

3. Define a new method called `sum_even_positions(self)` that returns the sum of the elements of a linked list at the even positions in the list. Node positions begin at 0. Return 0 if the list is empty.

   ANS:

```
# sum_even_positions(self):
def sum_even_positions(self):
    total = 0
    pos = 0
    current = self._head
    while current != None:
        if pos % 2 == 0:
            total += current._value
        current = current._next
        pos += 1

    return total
```

4. Below is the code for `find(self, item)`, which returns `True` if item is an element of the linked list and `False` otherwise:

```
def find(self, item):
    current = self._head
    while current != None:
        if item == current._value:
            return True
        else:
            return False

        current = current._next
```

   a) There is a common logic error in this code. What is the bug?

   ANS:

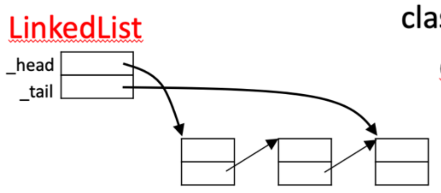
The if/else in the while loop will guarantee that that the loop is executed only once, and only one element of the list will be seen.

b) Rewrite the code with the bug fixed:

ANS:

```
def find(self, item):
    current = self._head
    while current != None:
        if item == current._value:
            return True
        current = current._next
    # if we get here, item was not found in the list
    return False
```

5. Suppose that we modify the linked list class to maintain a tail reference. The modified code for the `LinkedList` class and a sample linked list are shown below:



```
class LinkedList:
  def __init__(self):
     self._head = None
     self._tail = None
```

Write the `append(self, new)` method for the class.

ANS:

```
def append(self,new):
    if self._head == None:
        self._head = new
        self._tail = new
     else:
        self._tail._next = new

        self._tail = new
```