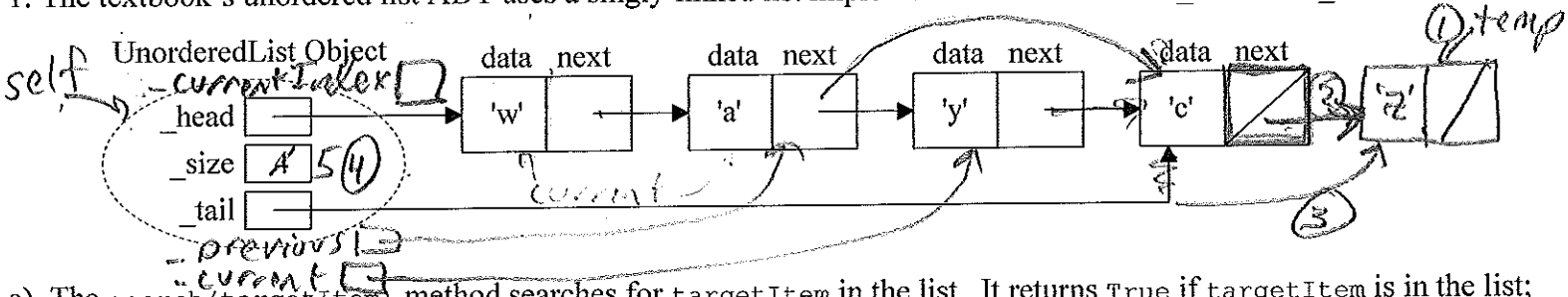
Data Structures (CS 1520) Lecture 8 Name:_____

1. The textbook's unordered list ADT uses a singly-linked list implementation. I added the `_size` and `_tail` attributes:



a) The `search(targetItem)` method searches for `targetItem` in the list. It returns `True` if `targetItem` is in the list; otherwise it returns `False`. Complete the `search(targetItem)` method code:

```
class UnorderedList:

    def search(self, targetItem):
        if self._current != None and self._current.getData() == targetItem:
            self._currentIndex = 0
            return True
        self._previous = None
        self._current = self._head
        while self._current != None:
            if self._current.getData() == targetItem:
                return True
            else:
                self._previous = self._current
                self._current = self._current.getNext()
                self._currentIndex += 1
        return False
```

b) The textbook's unordered list ADT **does not** allow duplicate items, so operations `add(item)`, `append(item)`, and `insert(pos, item)` would have what precondition?

Precondition: item is not in the list

c) Complete the `append(item)` method including a check of it's precondition(s)?

```
    def append(self, item):
        if self.search(item):
            raise ValueError("Cannot append duplicate item")
        temp = Node(item)
        if self._size == 0:
            self._head = temp
        else:
            self._tail.setNext(temp)
        self._tail = temp
        self._size += 1
```

d) Why do you suppose I added a `_tail` attribute? to get O(1) append

e) The textbook's `remove(item)` and `index(item)` operations "Assume the item is present in the list." Thus, they would have a precondition like "Item is in the list." When writing a program using an UnorderedList object (say `myGroceryList = UnorderedList()`), how would the programmer check if the precondition is satisfied?

```
itemToRemove = input("Enter the item to remove from the Grocery list: ")
if myGroceryList.search(itemToRemove):
    myGroceryList.remove(itemToRemove)
```

f) The `remove(item)` and `index(item)` methods both need to look for the `item`. What is inefficient in this whole process? Call search as user of unorderedList, method calls search to check precondition, and method has loop to look down list

g) Modify the `search(targetItem)` method code in (a) to set additional data attributes to aid the implementation of the `remove(item)` and `index(item)` methods.

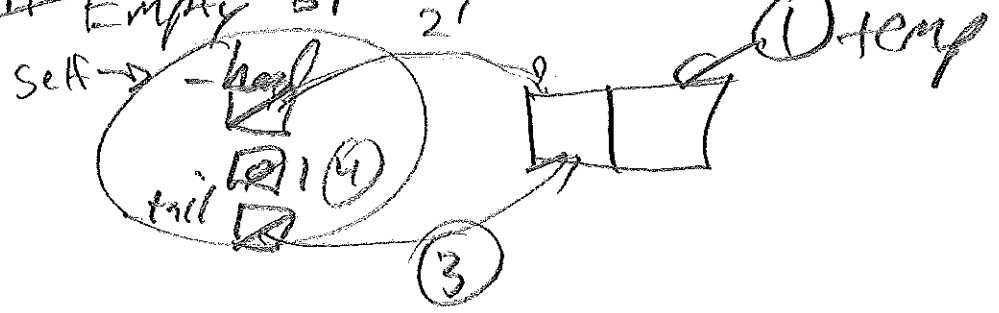
h) Write the `index(item)` method including a check of its precondition(s).

```
def index(self, item):
    if not self.search(item):
        raise ValueError("Cannot not call index if item not in list")
    return self._currentIndex
```

i) Write the `remove(item)` method including a check of its precondition(s).

```
def remove(self, item):
    if not self.search(item):
        raise ValueError("Cannot remove item not in list")
(0) temp = self._current
(1*) if self._current == self._head:
        self._head = self._current.getNext()
(1) else:
        self._previous.setNext(self._current.getNext())
(2) self._size -= 1
(2.5) if self._current == self._tail:
        self._tail = self._previous
(3) self._current = None
(4) return temp.getData()
```

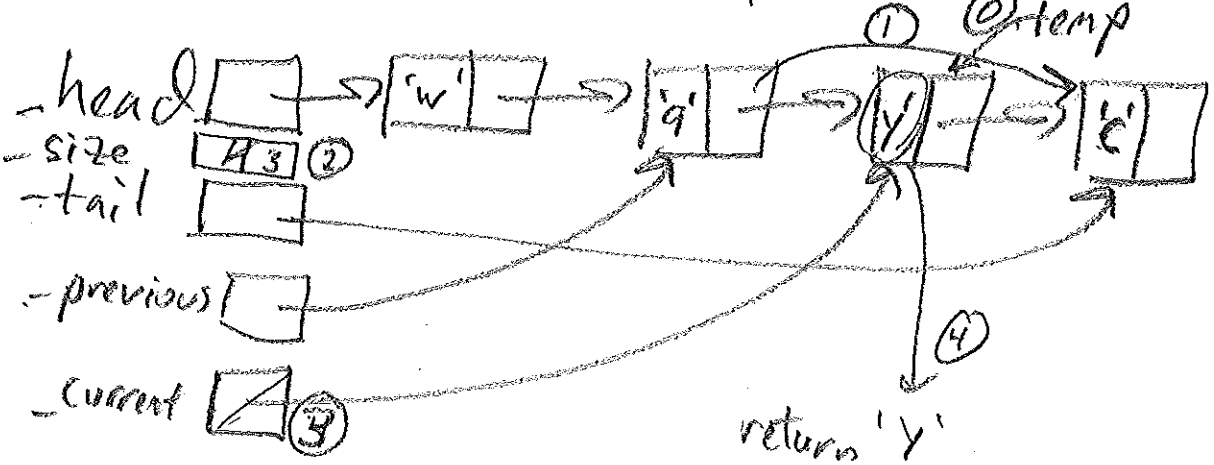
Append: Empty & special case



---

and return middle of list

Remove "normal" case picture; remove 'y'



0. temp = self._current
1. self._previous.setNext(self._current.getNext())
2. self._size -= 1
3. self._current = None
4. return temp.getData()

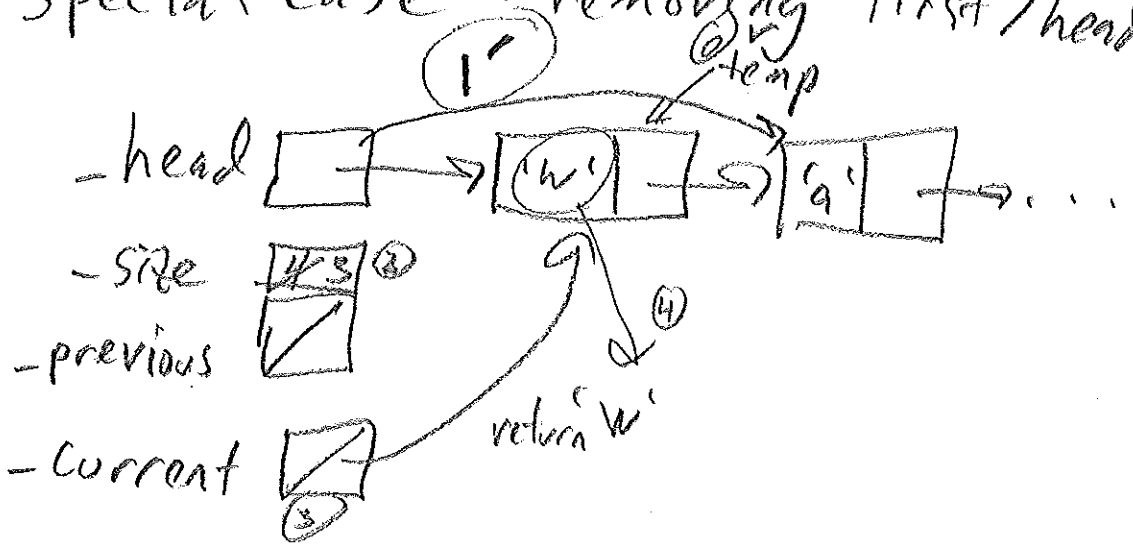
---

special cases:

- removing first item in list (head)
- removing tail item in list
- remove only item in list

Remove

Special case - removing first/head item, 'w'

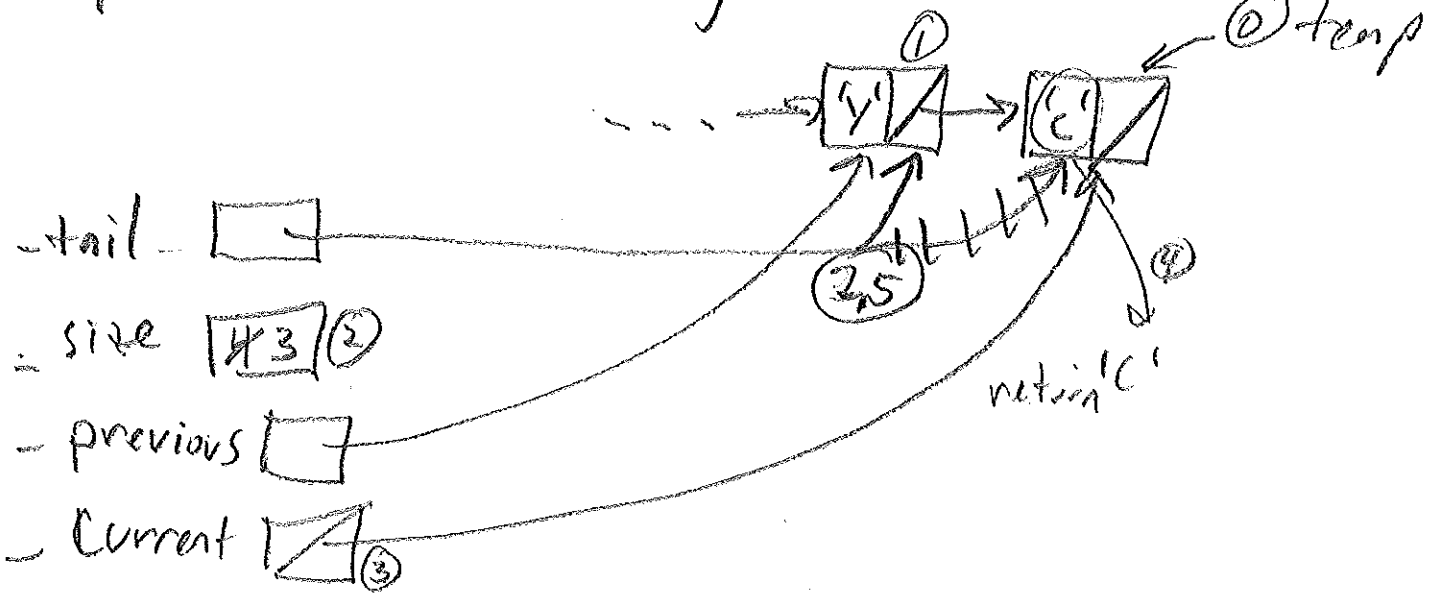


Note: normal case code step (1) cause error since self._previous has value of None

Alternate step (1') needed to set self._head around first node

---

special case removing tail item: 'c'



Note: normal case code steps all work, except _tail needs to be reset to the _previous as step (2.5)

Remove (and return) code – see Lecture 8 page 2

Claim is code on Lecture 8 page 2 works for removing only item in list.