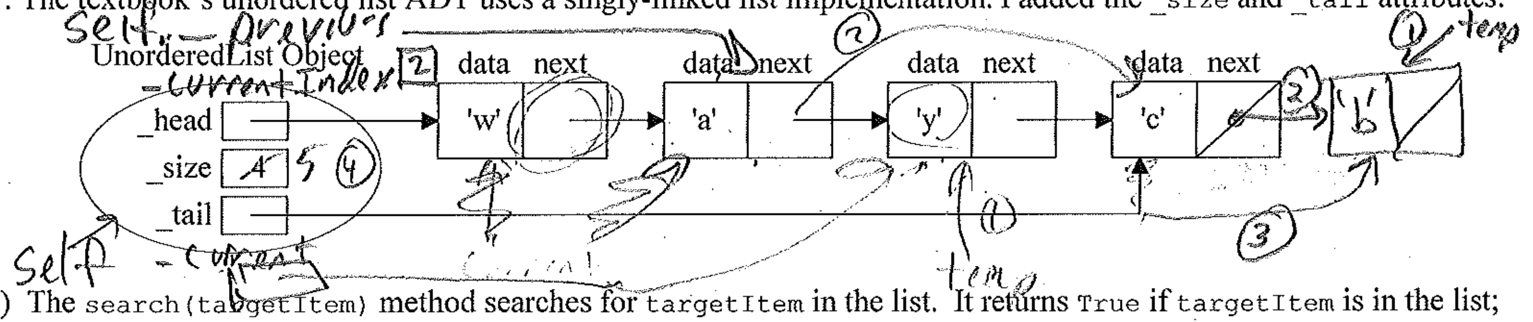
index: 0 1 2 3

Data Structures (CS 1520) Lecture 8 Name: Mark F.

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

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?

item is not in 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 items")
     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?

So we can find tail node in O(1)

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)
```

(1)

f) The `remove(item)` and `index(item)` methods both need to look for the `item`. What is inefficient in this whole process?

check precondition

```
def remove(self, item):
    if self.search(item) == False:
        raise ValueError("Cannot remove item not inlist")
```

(2) Two back-to-back searches, so modify search so second is fast.

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. ← NOTE: added self._currentIndex counter to search method

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

```
def index(self, item):
    if self.search(item) == False:
        raise - - -
    return self._currentIndex
```

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

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