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

```python
class UnorderedList:
    def __init__(self):
        self._head = None
        self._size = 0
        self._current = self._head

    def search(self, targetItem):
        if self._current is None:
            return False
        self._current_index = 0
        self._previous = None
        self._current = self._head
        while self._current != None:
            if self._current.data == targetItem:
                return True
            else:
                self._previous = self._current
                self._current = self._current.next
                self._current_index += 1
```

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:

```python
    if self._current.data == targetItem:
        return True
    else:
        self._previous = self._current
        self._current = self._current.next
        self._current_index += 1
```

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 its precondition(s):

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

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?

```python
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).

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

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

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

Remove "normal" case picture: remove 'y'

0. \( \text{temp} = \text{self.\_current} \)
1. \( \text{self.\_previous.\_setNext}(\text{self.\_current.\_getNext()}) \)
2. \( \text{self.\_size} -= 1 \)
3. \( \text{self.\_current} = \text{None} \)
4. return \( \text{temp.\_get\_Data()} \)

Special cases:
- removing first item in list
- removing tail item in list
- remove only item in list
Remove special case - removing first head item, w.

- head
- size 4
- previous
- current

Note: normal case code step 0 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 works except _tail needs to be reset to the _previous as step 2.5
Claim is code on Lecture 8 page 2

Works for removing only item

in list.

Return item.