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

```
UnorderedList Object
  \_head    \_size    \_tail
    'w'     4       
    'a'     
    'y'     
    'c'     
```

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

```python
class UnorderedList:
    
def search(self, targetItem):
```

- b) The textbook’s unordered list ADT \textbf{does not} allow duplicate items, so operations \texttt{add(item)}, \texttt{append(item)}, and \texttt{insert(pos, item)} would have what precondition?

- c) Complete the \texttt{append(item)} method including a check of it’s precondition(s)?

```python
    
def append(self, item):
```

- d) Why do you suppose I added a \_tail attribute?
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
e itemToRemove = input("Enter the item to remove from the Grocery list: ")
if
    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?

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):
```

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

```python
def remove(self, item):
```