1. The textbook's ordered list ADT uses a singly-linked list implementation. I added the _size, _tail, _current, _previous, and _currentIndex attributes:

The `search(targetItem)` method searches for `targetItem` in the list. It returns `True` if `targetItem` is in the list; otherwise it returns `False`. Additionally, it has the side-effects of setting _current, _previous, and _currentIndex. The complete `search(targetItem)` method code for the `OrderedList` is:

```python
class OrderedList:
    def search(self, targetItem):
        if self._current != None and self._current.getData() == targetItem:
            return True

        self._previous = None
        self._current = self._head
        self._currentIndex = 0
        while self._current != None:
            if self._current.getData() == targetItem:
                return True
            elif self._current.getData() > targetItem:
                return False
            else:  # inch-worm down list
                self._previous = self._current
                self._current = self._current.getNext()
                self._currentIndex += 1
        return False
```

a) What's the purpose of the "elif self._current.getData() > targetItem:" check?

b) Complete the `add(item)` method including a check of it's precondition: `newItem` is not in the list.

```python
def add(self, newItem):
    if self.search(newItem):
        raise ValueError("Cannot.."
    temp = Node(newItem)
    if self._previous == None:  # add at head
        self._head = temp
    else:
        self._previous.setNext(temp)
        temp.setNext(self._current)
    if self._current == None:  # add at tail
        self._tail = temp
    self._size += 1
```
2. A recursive function is one that calls itself. Complete the recursive code for the countDown function that is passed a starting value and proceeds to count down to zero and prints "Blast Off!!!".

Hint: The countDown function, like most recursive functions, solves a problem by splitting the problem into one or more simpler problems of the same type. For example, countDown(10) prints the first value (i.e., 10) and then solves the simpler problem of counting down from 9. To prevent "infinite recursion", if-statement(s) are used to check for trivial base case(s) of the problem that can be solved without recursion. Here, when we reach a countDown(0) problem we can just print "Blast Off!!!".

```python
### File: countDown.py ###

def main():
    start = eval(input("Enter count down start: "))
    print("\nCount Down:"))
    countDown(start)

def countDown(count):
    if count <= 0:
        print('Blast Off!!!')
    else:
        print(count)
        countDown(count - 1)

main()
```

Program Output:

<table>
<thead>
<tr>
<th>Enter count down start: 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count Down:</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>Blast Off!!!</td>
</tr>
</tbody>
</table>

a) Trace the function call countDown(5) on paper by drawing the run-time stack and showing the output.

(see attached)

b) What do you think will happen if your call countDown(-1)? "infinite recursion"

c) Why is there a limit on the depth of recursion? run out of memory
Call function/method

- push call-frame on stack
  - return addr - where to continue after returning/end function
  - parameters
  - local variables

Blast off!!!

import sys
sys.setrecursionlimit(10000)
3. Complete the recursive `strHelper` function in the `__str__` method for our `OrderedList` class.

```python
def __str__(self):
    """Returns a string representation of the list with a space between each item. """
    def strHelper(current):
        if current == None:
            return 
        else:
            return str(current.getData()) + " " + strHelper(current.getNext())

    return "(head) " + strHelper(self._head) + "(tail)"
```

4. Some mathematical concepts are defined by recursive definitions. One example is the Fibonacci series:

\[
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots
\]

After the second number, each number in the series is the sum of the two previous numbers. The Fibonacci series can be defined recursively as:

- \( F_0 = 0 \)
- \( F_1 = 1 \)
- \( F_n = F_{n-1} + F_{n-2} \) for \( n \geq 2 \).

a) Complete the recursive function:

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

b) Draw the call tree for `fib(5)`.

---

Lecture 9 - Page 3