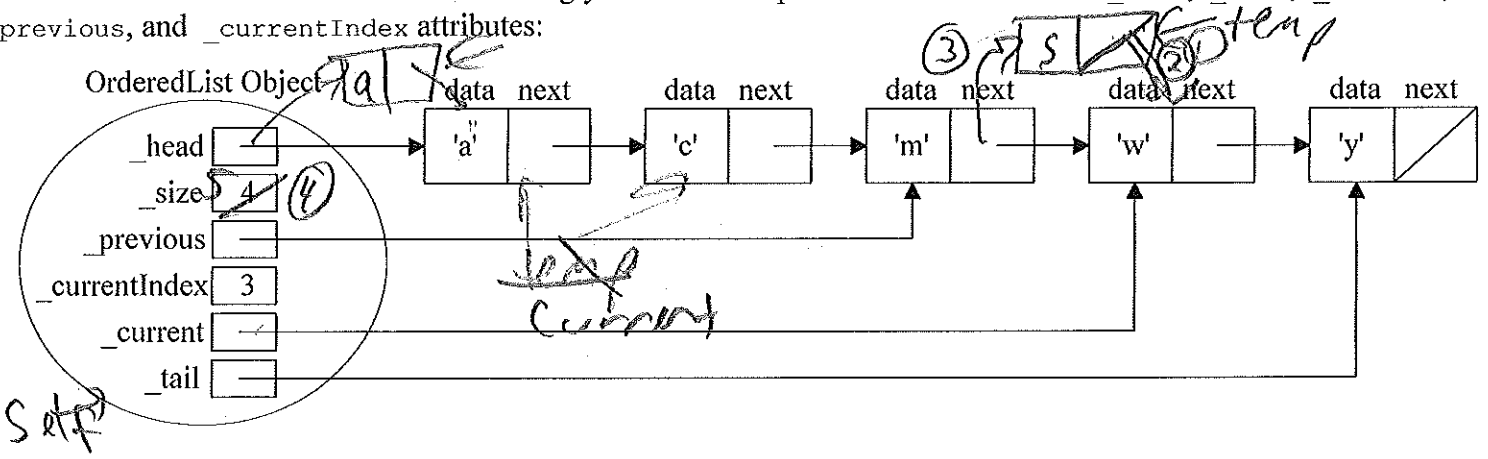
Data Structures (CS 1520) Lecture 9 Name: Mark F.

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:

```
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
```

temp = Node(newItem)
temp.setNext(self._current)
self._previous.setNext(temp)

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

Since they are ordered items, we would have found it already.

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

```
def add(self, newItem):
    if self.search(newItem):
        raise ValueError("Cannot add duplicates to ordered list.")

    temp = Node(newItem)
    temp.setNext(self._current)
    if self._previous == None:
        self._head = temp
    else:
        self._previous.setNext(temp)
    self._size += 1
    if self._current == None:
        self._tail = temp
    self._current = None
```

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!!!".

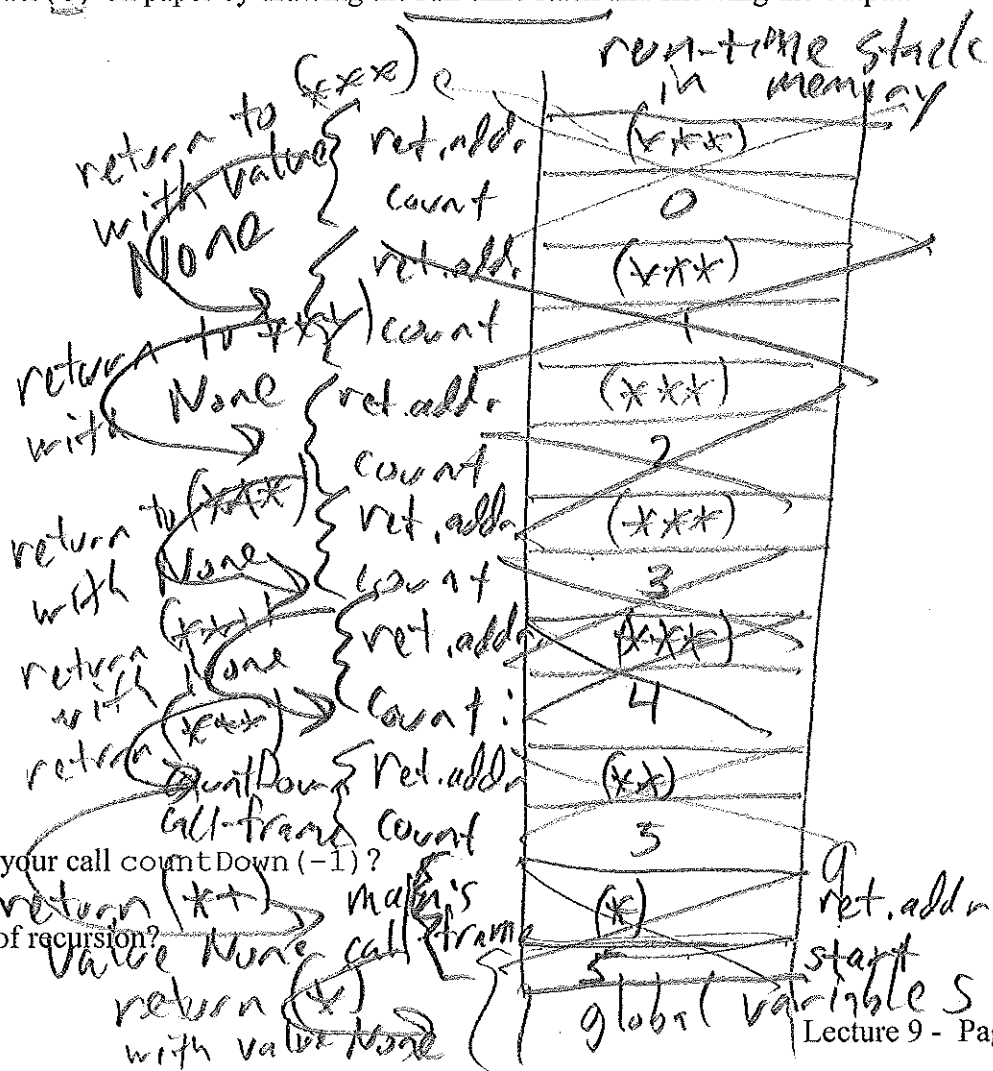
Hint: The `countDown` function, like most recursive functions, solves a problem by splitting the problem into one or more simpler problems of the same type. For example, `countDown(10)` prints the first value (i.e, 10) and then solves the simpler problem of counting down from 9. To prevent "infinite 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!!!".

```
""" 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:

```
Enter count down start: 10

Count Down:
10
9
8
7
6
5
4
3
2
1
Blast Off!!!
```

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



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?

memory is limited so run-time stack is too.

(Method)

Function call - cause call-frame to be pushed on the run-time stack

call-frame has three things
- return address - where to return after function ends
- parameters
- local variables - new variables defined in the function

When function ends or "returns", we pop the call-frame and return execution at the return address. If no value is returned, the default is the None value.

3. Complete the recursive `strHelper` function in the `__str__` method for our `OrderedList` class.

```
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 defining by recursive definitions. One example is the Fibonacci series:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

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:

$Fib_0 = 0$

$Fib_1 = 1$

$Fib_N = Fib_{N-1} + Fib_{N-2}$ for $N \geq 2$.

a) Complete the recursive function: `def fib (n):`

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

Non-recursive str for ordered list

```
temp = self._head
resultStr = ""
while temp != None:
    resultStr += str(temp.getData()) + " "
    temp = temp.getNext()
return resultStr
```