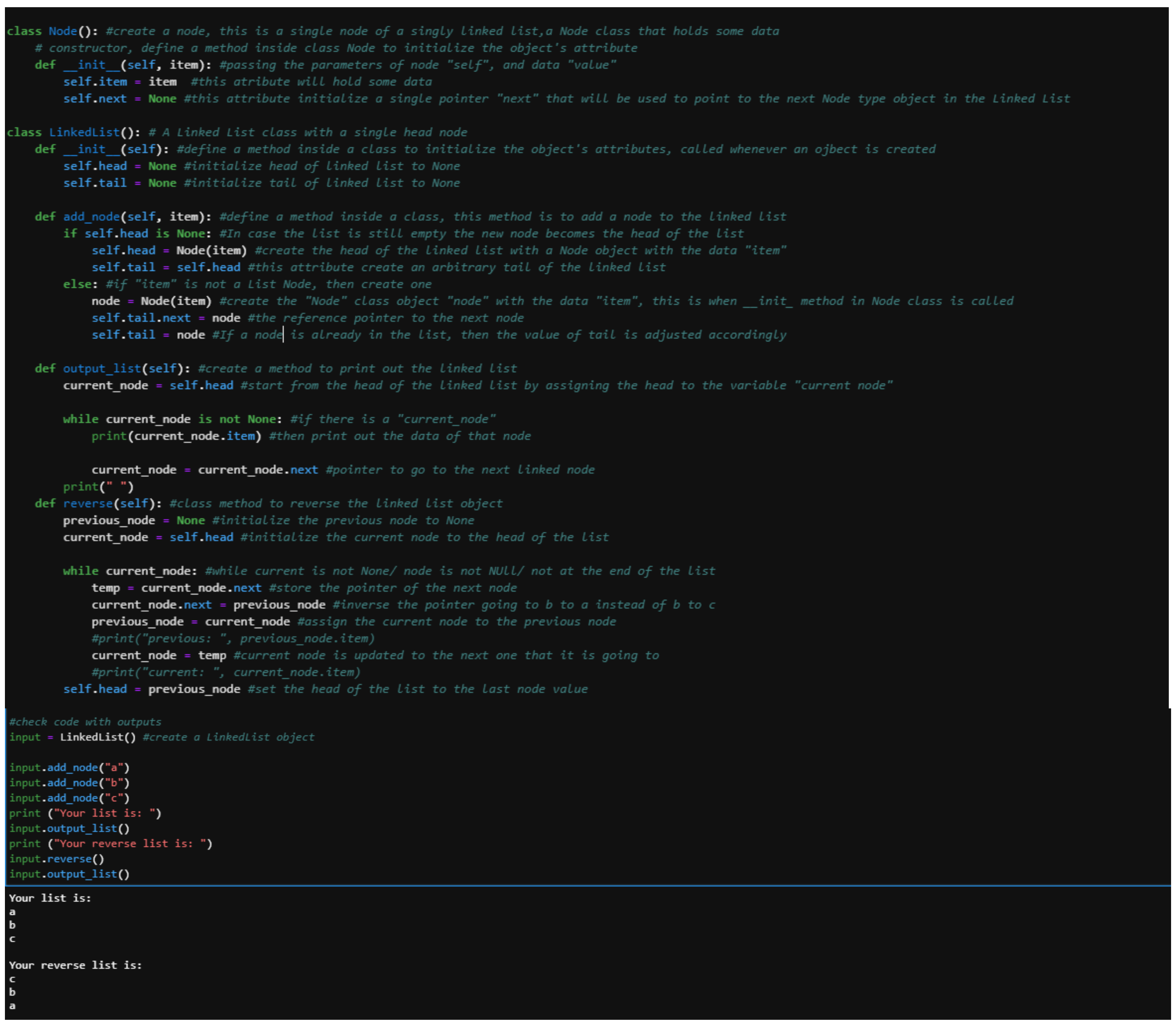
Practice M3

Saturday, October 9, 2021 10:27 PM

Phuc Tran (Jennifer)
MSWE - Fall 2021
241R - Applied Data Structures & Algorithms
The Algorithms Design Manual - Second Edition
Homework: 3-2 3-18 3-20

3-2 Write a program to reverse the direction of a given singly-linked list. In other words, after the reversal, all pointers should now point backwards. Your algorithm should take linear time.

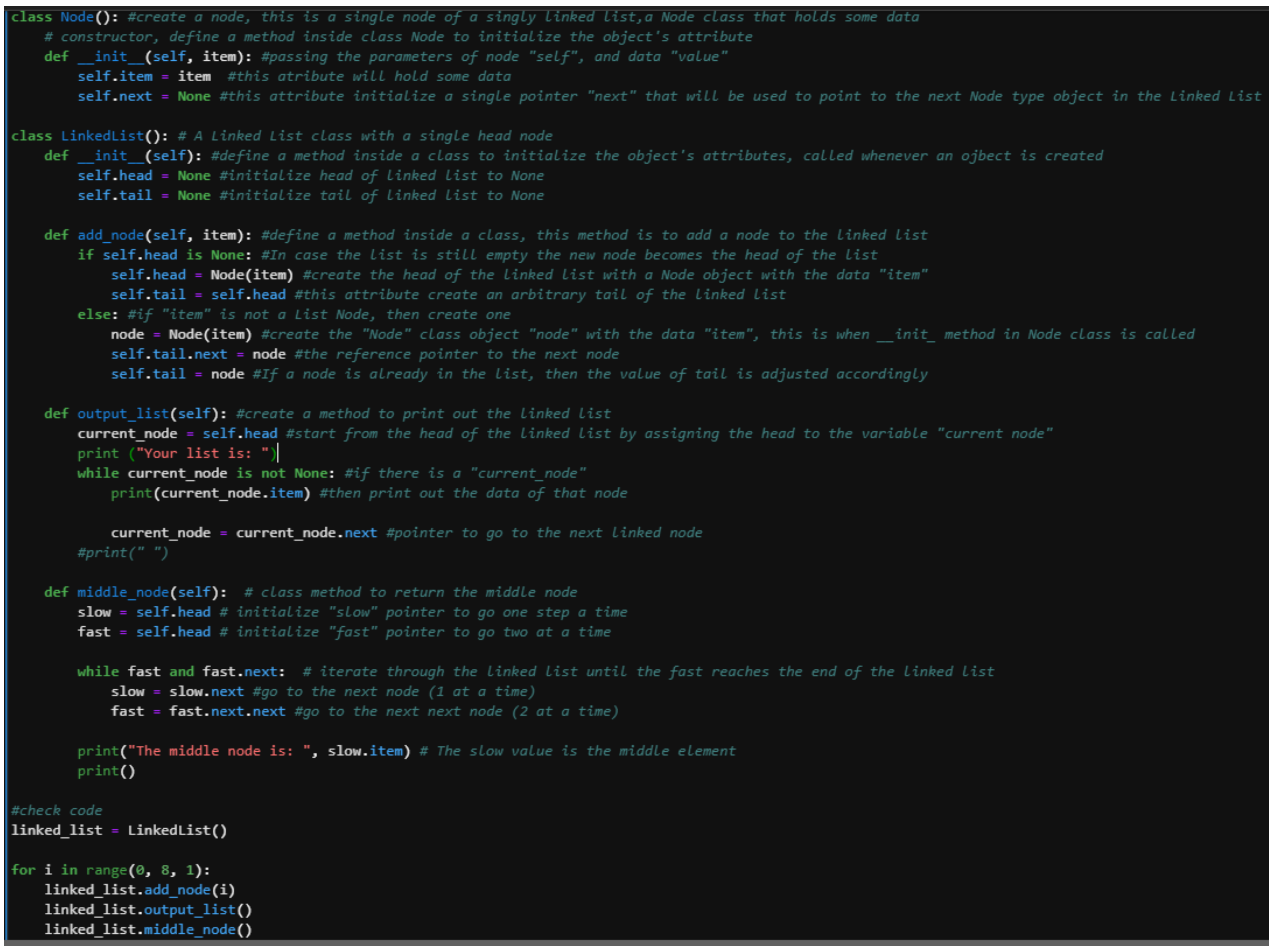


```
class Node(): #create a node, this is a single node of a singly linked list,a Node class that holds some data
    # constructor, define a method inside class Node to initialize the object's attribute
    def __init__(self, item): #passing the parameters of node "self", and data "value"
        self.item = item  #this atribute will hold some data
        self.next = None #this attribute initialize a single pointer "next" that will be used to point to the next Node type object in the Linked List

class LinkedList(): # A Linked List class with a single head node
    def __init__(self): #define a method inside a class to initialize the object's attributes, called whenever an ojbect is created
        self.head = None #initialize head of linked list to None
        self.tail = None #initialize tail of linked list to None

    def add_node(self, item): #define a method inside a class, this method is to add a node to the linked list
        if self.head is None: #In case the list is still empty the new node becomes the head of the list
            self.head = Node(item) #create the head of the linked list with a Node object with the data "item"
            self.tail = self.head #this attribute create an arbitrary tail of the linked list
        else: #if "item" is not a List Node, then create one
            node = Node(item) #create the "Node" class object "node" with the data "item", this is when __init_ method in Node class is called
            self.tail.next = node #the reference pointer to the next node
            self.tail = node #If a node is already in the list, then the value of tail is adjusted accordingly

    def output_list(self): #create a method to print out the linked list
        current_node = self.head #start from the head of the linked list by assigning the head to the variable "current node"

        while current_node is not None: #if there is a "current_node"
            print(current_node.item) #then print out the data of that node

            current_node = current_node.next #pointer to go to the next linked node
        print(" ")
    def reverse(self): #class method to reverse the linked list object
        previous_node = None #initialize the previous node to None
        current_node = self.head #initialize the current node to the head of the list

        while current_node: #while current is not None/ node is not NULL/ not at the end of the list
            temp = current_node.next #store the pointer of the next node
            current_node.next = previous_node #inverse the pointer going to b to a instead of b to c
            previous_node = current_node #assign the current node to the previous node
            #print("previous: ", previous_node.item)
            current_node = temp #current node is updated to the next one that it is going to
            #print("current: ", current_node.item)
        self.head = previous_node #set the head of the list to the last node value

#check code with outputs
input = LinkedList() #create a LinkedList object

input.add_node("a")
input.add_node("b")
input.add_node("c")
print ("Your list is: ")
input.output_list()
print ("Your reverse list is: ")
input.reverse()
input.output_list()
```

```
Your list is:
a
b
c

Your reverse list is:
c
b
a
```

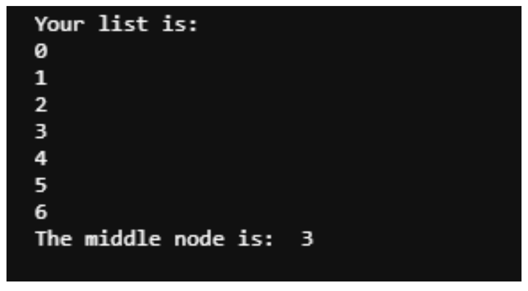
3-18 What method would you use to look up a word in a dictionary?
Binary search.
Divide the file into halves
Jump into the midpoint locations
Keep doing so until the word is found.

3-20 Write a function to find the middle node of a singly-linked list.



```
class Node(): #create a node, this is a single node of a singly linked list,a Node class that holds some data
    # constructor, define a method inside class Node to initialize the object's attribute
    def __init__(self, item): #passing the parameters of node "self", and data "value"
        self.item = item  #this atribute will hold some data
        self.next = None #this attribute initialize a single pointer "next" that will be used to point to the next Node type object in the Linked List

class LinkedList(): # A Linked List class with a single head node
    def __init__(self): #define a method inside a class to initialize the object's attributes, called whenever an ojbect is created
        self.head = None #initialize head of linked list to None
        self.tail = None #initialize tail of linked list to None

    def add_node(self, item): #define a method inside a class, this method is to add a node to the linked list
        if self.head is None: #In case the list is still empty the new node becomes the head of the list
            self.head = Node(item) #create the head of the linked list with a Node object with the data "item"
            self.tail = self.head #this attribute create an arbitrary tail of the linked list
        else: #if "item" is not a List Node, then create one
            node = Node(item) #create the "Node" class object "node" with the data "item", this is when __init_ method in Node class is called
            self.tail.next = node #the reference pointer to the next node
            self.tail = node #If a node is already in the list, then the value of tail is adjusted accordingly

    def output_list(self): #create a method to print out the linked list
        current_node = self.head #start from the head of the linked list by assigning the head to the variable "current node"
        print ("Your list is: ")
        while current_node is not None: #if there is a "current_node"
            print(current_node.item) #then print out the data of that node

            current_node = current_node.next #pointer to go to the next linked node
        #print(" ")

    def middle_node(self):  # class method to return the middle node
        slow = self.head # initialize "slow" pointer to go one step a time
        fast = self.head # initialize "fast" pointer to go two at a time

        while fast and fast.next:  # iterate through the linked list until the fast reaches the end of the linked list
            slow = slow.next #go to the next node (1 at a time)
            fast = fast.next.next #go to the next next node (2 at a time)

        print("The middle node is: ", slow.item) # The slow value is the middle element
        print()

#check code
linked_list = LinkedList()

for i in range(0, 8, 1):
    linked_list.add_node(i)
    linked_list.output_list()
    linked_list.middle_node()
```

Output:



```
Your list is:
0
1
2
3
4
5
6
The middle node is:  3
```