CSC 212 Data Structures

Exercise 4 – Linked Lists

To construct a linked list, we utilize 2 different class definitions: a Node class and a List class. Let’s first consider the Node class:

class Node:

def \_\_init\_\_(self, initdata):

self.data = initdata

self.next = None

def getData(self):

return self.data

def getNext(self):

return self.next

def setData(self, newdata):

self.data = newdata

def setNext(self, newnext):

self.next = newnext

Questions:

1. Briefly explain the purpose of the \_\_init\_\_ method. When is this method invoked? Briefly explain the 2 assignment statements in this method.

Init tells the function where to start and gives the minimum info required to run successfully. It runs automatically when the code is run.

1. There are 2 groups of methods: the get methods (accessors) and the set methods (mutators). Briefly explain the role of each, the parameter and the return value when appropriate.

The difference is that they are used to edit the values that are set in the init function when an instance is created. The setData allows you to add new data to replace the data from when the instance was created. The set next changes what is next in line so that way every item is pointed to the next item in line.

For this exercise, we will consider the Unordered List as our list class.

class UnorderedList:

def \_\_init\_\_(self):

self.head = None

def isEmpty(self):

return self.head == None

def add(self,item):

temp = Node(item)

temp.setNext(self.head)

self.head = temp

def size(self):

current = self.head

count = 0

while current != None:

count = count + 1

current = current.getNext()

return count

def search(self,item):

current = self.head

found = False

while current != None and not found:

if current.getData() == item:

found = True

else:

current = current.getNext()

return found

def remove(self,item):

current = self.head

previous = None

found = False

while not found:

if current.getData() == item:

found = True

else:

previous = current

current = current.getNext()

if previous == None:

self.head = current.getNext()

else:

previous.setNext(current.getNext())

mylist = UnorderedList() #1

mylist.add(31) #2

mylist.add(77) #3

mylist.add(17) #4

mylist.add(93) #5

mylist.add(26) #6

mylist.add(54) #7

print(mylist.size()) #8

print(mylist.search(93)) #9

print(mylist.search(100)) #10

mylist.add(100) #11

print(mylist.search(100)) #12

print(mylist.size()) #13

mylist.remove(54) #14

print(mylist.size()) #15

mylist.remove(93) #16

print(mylist.size()) #17

mylist.remove(31) #18

print(mylist.size()) #19

print(mylist.search(93)) #20

Questions:

1. Referring to the 20 instructions after the Unordered List class definition, please answer the following questions. You will provide an answer for each of the 20 instructions. You might want to use a separate sheet for your answers. Hand drawn diagrams are acceptable. For each drawing, you must indicate the *head* pointer and the *None* field indicating the end of the list.
   1. What is accomplished in instruction #1? We make a new instance of an unordered list
   2. Draw the content of the linked list after each *add* and *remove* instruction
   3. Trace the code and give the output as described in the code

1. Develop an *append* method to add a new node to the end of the list (much like the add method, except it adds the new node to the end of the list).

def append(self,item):

current = self.head

count = 0

while current != None:

count = count + 1

current = current.getNext()

temp = Node(item)

temp.setNext(count+1)

self.head = temp

I have no clue how to test where in the list that was added but it was definitely added.