 **Northwestern Polytechnic University**

**Python Programming**

**Homework Assignment #5**

**Due day: 11/10/2021**

**Instruction:**

1. **Push the source code to GitHub or answer sheet in word file**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework submission could not be accepted.**

**4. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

1. Write a function to check if the element exists or not in the linked list.

def cntn\_link(s, elm):

"""Return True if elm is in the linked list s

>>> cntn\_link (link(1, link(2, link(3, empty))), 1)

True

>>> cntn\_link (link(1, link(2, link(3, empty))), 4)

False

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def cntn\_link(s, elm):

while s != empty:

if first(s)==elm:

return True

else:

s = rest(s)

cntn\_link(s,elm)

return False

cntn\_link (link(1, link(2, link(3, empty))), 1)

cntn\_link (link(1, link(2, link(3, empty))), 4)

1. Create a function to print linked list as follows.

def prnt\_lnk(s):

"""

>>> prnt\_lnk ( link(1, link(2, link(3, link(4, empty)))) )

<1 2 3 4>

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def prnt\_lnk(s):

line = ''

while s != empty:

if line:

line += ' '

line += str(first(s))

s = rest(s)

print(line)

prnt\_lnk( link(1, link(2, link(3, link(4,empty)))))

1. Implement a function to create a new linked list in the reverse order.

def rvrs\_lnk(s):

"""Return linked list reversed

>>> rvrs\_lnk (link(1, link(2, link(3, link(4, empty)))))

[4, [3, [2, [1, [ ] ]]]]

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def prnt\_lnk(s):

line = ''

while s != empty:

if line:

line += ' '

line += str(first(s))

s = rest(s)

print(line)

def rvrs\_lnk(s):

line = ''

while s != empty:

line += str(first(s))

s = rest(s)

line=list(line)

line.reverse()

return link(line[0],link(line[1],link(line[2],link(line[3],empty))))

rvrs\_lnk(link(1, link(2, link(3, link(4, empty)))))

1. Write a function srt (lnk) function, which returns True if the linked list lnk is sorted ascendingly from the left to right. If two adjacent elements are equal, the linked list can still be considered sorted.

def srt (lnk):

""" if the linked list lnk is sorted, then return True

>>> lnk1 = link(1, link(2, link(3, link(4,empty))))

>>> srt (lnk1)

True

>>> lnk2 = link(1, link(3, link(2, link(4, link(5, empty)))))

>>> srt (lnk2)

False

>>> lnk3 = link(3, link(3, link(3, empty)))

>>> srt (lnk3)

True

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def srt (lnk):

i=0

while lnk != empty:

x=first(lnk)

if(x <=first(lnk)):

i=1

else:

break

lnk=rest(lnk)

if i>1:

return False

else:

return True

lnk1 = link(1, link(2, link(3, link(4,empty))))

srt (lnk1)

lnk2 = link(1, link(3, link(2, link(4, link(5, empty)))))

srt (lnk2)

lnk3 = link(3, link(3, link(3, empty)))

srt (lnk3)

1. Write a function with arguments a linked list lnk and a function g, which is applied to each number in lnk and returns the sum. If the linked list is empty, the sum is 0.

def sum\_lnk(lnk, g):

"""Applies a function g to each element in lnk and returns the sum

of them

>>> sqr = lambda x: x \* x

>>> dbl = lambda y: 2 \* y

>>> lnk1 = link(1, link(2, link(3, link(4, empty))))

>>> sum\_lnk (lnk1, sqr)

30 # sqr(1) + sqr(2) + sqr(3) + sqr(4)

>>> lnk2 = link(3, link(5, link(4, link(6, empty))))

>>> sum\_lnk (lnk2, dbl)

36 # dbl(3)+ dbl(5)+ dbl(4)+ dbl(6)

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def sum\_lnk (lnk, f):

sum=0

while lnk != empty:

if sum:

sum +=0

sum += f(first(lnk))

lnk = rest(lnk)

print(sum)

sqr = lambda x: x \* x

dbl = lambda y: 2 \* y

lnk1 = link(1, link(2, link(3, link(4, empty))))

sum\_lnk (lnk1, sqr)

lnk2 = link(3, link(5, link(4, link(6, empty))))

sum\_lnk (lnk2, dbl)

1. Define a function with input parameters a linked list, lnk, and two elements, u & v. The function returns linked list but with all elements of u substituted with v.

def change(lnk, u, v):

"""Returns a linked list matching lnk but with all elements of u replaced by v. If u does not appear in lnk, then return lnk

>>> l = link(1, link(2, link(3, empty)))

>>> n=change(l, 3, 1)

>>> n

[1, [2, [1, [ ] ]]]

>>> m=change(n, 1, 2)

>>> m

[2, [2, [2, [ ]]]]

>>> change(m, 5, 1)

[2, [2, [2, [ ]]]]

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def prnt\_lnk(s):

line = ''

while s != empty:

if line:

line += ' '

line += str(first(s))

s = rest(s)

print(line)

def apnd(s, elem):

if s[1] == 'empty':

s[1] = [elem, empty]

else:

apnd(s[1], elem)

return s

def change(lnk, u, v):

new=empty

while lnk != empty:

if first(lnk)!=u:

new= link(first(lnk),new)

elif first(lnk)==u:

apnd(new,v)

lnk = rest(lnk)

return new

l = link(1, link(2, link(3, empty)))

n=change(l, 3, 1)

print(n)

m=change(n, 1, 2)

print(m)

change(m, 5, 1)

1. Generate a function to append element to the end of linked list.

def apnd(lnk, m):

"""Adds the element m to the end of lnk  
  
 >>> l = link(1, link(2, link(3, empty)))  
 >>> n = apnd (l, 4) # n = [1, [2, [3, [4, [] ]]]]  
 >>> first(rest(rest(rest(n))))  
 4  
 """

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def apnd(s, elem):

if s[1] == 'empty':

s[1] = [elem, empty]

else:

apnd(s[1], elem)

return s

l = link(1, link(2, link(3, empty)))

n = apnd (l, 4)

first(rest(rest(rest(n))))

1. Implement the insert function that creates a copy of the original list with an item inserted at the specific index. If the index is greater than the current length, you should insert the item at the end of the list.

def insrt(l, elm, ind):

"""

>>> l = link(11, link(12, link(13, empty)))

>>> n = insrt (l, 2021, 1)

>>> n

[11, [2021, [12, [13, [ ] ]]]]

>>> m = insrt(n, 2022, 20)

>>> m

[11 [2021 [12 [13 [2022, [ ] ]]]]]

"""

empty = 'empty'

def is\_link(s):

return s == empty or (type(s) == list and len(s) == 2 and is\_link(s[1]))

def link(first, rest=empty):

assert is\_link(rest), 'rest must be a linked list.'

return [first, rest]

def first(s):

assert is\_link(s), 'first only applies to linked lists.'

assert s != empty, 'empty linked list has no first element.'

return s[0]

def rest(s):

assert is\_link(s), 'rest only applies to linked lists.'

assert s != empty, 'empty linked list has no rest.'

return s[1]

def apnd(s, elem):

if s[1] == 'empty':

s[1] = [elem, empty]

else:

apnd(s[1], elem)

return s

def insrt(l, elm, ind):

i=0

new = empty

if l[1] == 'empty':

l[1] = [elm, empty]

while i<ind:

new=link(first(l),new)

l=rest(l)

i+=1

if i == ind:

apnd(new, elm)

apnd(new,l)

return new

l = link(11, link(12, link(13, empty)))

n = insrt (l, 2021, 1)

print(n)