 **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 = []

def cntn\_link(s, elm):

if s == empty:

return False

elif fst(s) == elm:

return True

else:

return False or cntn\_link(rest(s), elm)

def lnk(s):

return s == empty or (len(s) == 2 and lnk(s[1]))

def fst(s):

assert lnk(s), "first for lnkdlnks."

assert s != empty,"no first"

return s[0]

def rest(s):

assert lnk(s), "rest only lnkdlnks"

assert s != empty,"no rest"

return s[1]

def link(fst, rest):

assert lnk(rest), "rest should be lnkd lst."

return [fst, rest]

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

print(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=[]

def prnt\_lnk(s):

ls = '<'

while s != empty:

ls += str(fst(s)) + ' '

s = rest(s)

ls += '>'

return ls

def lnk(s):

return s == empty or (len(s) == 2 and lnk(s[1]))

def fst(s):

assert lnk(s), "fst applies to lnked lst."

assert s != empty, "no fst"

return s[0]

def link(fst, rest):

assert lnk(rest), "rest should b linked list."

return [fst, rest]

def rest(s):

assert lnk(s), "rest only lnkd lsts."

assert s != empty, "no rest"

return s[1]

print(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, [ ] ]]]]*

*"""*

#todefextlink,lenlnk,lnk,link,fst,rest

empty=[]

def rvrs\_lnk(s):

res = empty

while lenlnk\_recur(s) > 0:

res = extdlnk(link(fst(s), empty), res)

s = rest(s)

return res

def lenlnk\_recur (s):

if s == empty:

return 0

return 1+ lenlnk\_recur (rest(s))

def extdlnk(s, t):

assert lnk(s) and lnk(t)

if s == empty:

return t

else:

return link(fst(s), extdlnk(rest(s), t))

def lnk(s):

return s == empty or (len(s) == 2 and lnk(s[1]))

def fst(s):

assert lnk(s), "fst lnkdlst."

assert s != empty, "no first"

return s[0]

def rest(s):

assert lnk(s), "rest only lnkdlst

assert s != empty, "no rest"

return s[1]

def link(fst, rest):

assert lnk(rest), "rest have to lnkdlst

return [fst, rest]

print(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=[]

def srt (lnk):

if lenlnk\_recur(lnk) <= 1:

return True

elif fst(lnk) > fst(rest(lnk)):

return False

else:

return True and srt(rest(lnk))

def lenlnk\_recur (s):

if s == empty:

return 0

return 1+ lenlnk\_recur (rest(s))

def link(fst, rest):

assert lns(rest), "rest lnkdlst

return [fst, rest]

def lns(s):

return s == empty or (len(s) == 2 and lns(s[1]))

def fst(s):

assert lns(s), "first only lnkdlst."

assert s != empty, "no first"

return s[0]

def rest(s):

assert lns(s), "rest only lnkd lst"

assert s != empty, "no rest"

return s[1]

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

print(srt (lnk1))

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

print(srt (lnk2))

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

print(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=[]

def sum\_lnk(lnk, g):

result = 0

while lenlnk\_recur (lnk) > 0:

result += g(fst(lnk))

lnk = rest(lnk)

return result

def lenlnk\_recur (s):

if s == empty:

return 0

return 1+ lenlnk\_recur (rest(s))

def link(fst, rest):

assert lns(rest), "rest lnkd lst

return [fst, rest]

def lns(s):

return s == empty or (len(s) == 2 and lns(s[1]))

def fst(s):

assert lns(s), "first only lnkdlst."

assert s != empty, "no first"

return s[0]

def rest(s):

assert lns(s), "rest only lnkd lst"

assert s != empty, "no rest"

return s[1]

sqr = lambda x: x \* x

dbl = lambda y: 2 \* y

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

print(sum\_lnk (lnk1, sqr) )

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

print(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=[]

def change(lnk, u, v):

result = empty

while lenlnk\_recur(lnk) > 0:

if fst(lnk) == u:

result = extdlnk(result, [v, []])

else:

result = extdlnk(result, [fst(lnk), []])

lnk = rest(lnk)

return result

def extdlnk(s, t):

assert lnk(s) and lnk(t)

if s == empty:

return t

else:

return link(fst(s), extdlnk(rest(s), t))

def lenlnk\_recur (s):

if s == empty:

return 0

return 1+ lenlnk\_recur (rest(s))

def fst(s):

assert lnk(s), "fst only for lnkdlst "

assert s != empty, "no first"

return s[0]

def link(fst, rest):

assert lnk(rest), "rest for lnkdlst"

return [fst, rest]

def rest(s):

assert lnk(s), "rest for lnkd lst"

assert s != empty, "no rest"

return s[1]

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=[]

def apnd(lnk, m):

return extdlnk(lnk, [m, []])

def fst(s):

assert lnk(s), "fst lnkd lst"

assert s != empty, "no first"

return s[0]

def link(fst, rest):

assert lnk(rest), "resthave to be lnkd lst"

return [fst, rest]

def rest(s):

assert lnk(s), "rest lnkdlst"

assert s != empty, "no rest"

return s[1]

def lnk(s):

return s == empty or (len(s) == 2 and lnk(s[1]))

def extdlnk(s, t):

assert lnk(s) and lnk(t)

if s == empty:

return t

else:

return link(fst(s), extdlnk(rest(s), t))

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

n = apnd (l, 4)

print(fst(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=[]

def insrt(l, elm, ind):

result = empty

i = 0

tmp = lenlnk\_recur(l)

while lenlnk\_recur(l) > 0:

if i == ind:

result = extdlnk(result, link(elm, empty))

i += 1

else:

result = extdlnk(result, link(fst(l), empty))

l = rest(l)

i += 1

if ind > i:

result = extdlnk(result, link(elm, empty))

return result

def lenlnk\_recur (s):

if s == empty:

return 0

return 1+ lenlnk\_recur (rest(s))

def extdlnk(s, t):

assert lnk(s) and lnk(t)

if s == empty:

return t

else:

return link(fst(s), extdlnk(rest(s), t))

def lnk(s):

return s == empty or (len(s) == 2 and lnk(s[1]))

def fst(s):

assert lnk(s), "first lnkdlst"

assert s != empty, "no first"

return s[0]

def rest(s):

assert lnk(s), "rest lnkdlst."

assert s != empty, "no rest"

return s[1]

def link(fst, rest):

assert lnk(rest), "rest haveto be lnkdlst"

return [fst, rest]

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

n = insrt(l, 2019, 1)

print(n)

m = insrt(n, 2020, 20)

print(m)