# Linked List Node

In [3]:

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
a=Node(13)
b=Node(23)
a.next=b
print(a.data)
print(b.data)
print(a.next.data)
print(a.next.next)
print(a.next)
print(b)
print(b.next)
```

```
13
23
23
None
<__main__.Node object at 0x000002128EB4F988>
<__main__.Node object at 0x000002128EB4F988>
None
```

# Quiz

In [4]:

```
#quiz-1
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def printLL(head):
    while head is not None:
        print(head.data,end=" ")
        head = head.next



node1 = Node(10)
node2 = Node(20)
node2.next = node1
printLL(node2)
```

```
20 10
```

In [5]:

```
# Quiz-2
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def printLL(head):
    while head is not None:
        print(head.data,end=" ")
        head = head.next



node1 = Node(10)
node2 = Node(20)
node3 = Node(30)
node4 = Node(40)
node1.next = node2
node2.next = node3
node3.next = node4
printLL(node2)
```

```
20 30 40
```

# Linked List Input - 1

In [14]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def takeInput():

    inputList=[int(ele) for ele in input().split()]
    head = None
    for currData in inputList:
        if currData==-1:
            break
        newNode = Node(currData)
        if head is None:
            head = newNode
        else:
            curr = head
            while curr.next is not None:
                curr=curr.next
            curr.next=newNode
    return head
```

# Print Linked List

In [15]:

```
def printLL(head):
    while head is not None:
        print(str(head.data)+"->",end="")
        head = head.next
    print("None")
    return
head=takeInput()
printLL(head)
```

```
1 2 3 4 5 6
1->2->3->4->5->6->None
```

# Time Complexity Of Taking Input

In [ ]:

```
#the time comlexity is o(n^2) now below code or linked list the time comlexity is o(n)
```

# Linked List Input - 2

In [17]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def takeInput():

    inputList=[int(ele) for ele in input().split()]
    head = None
    tail = None
    for currData in inputList:
        if currData==-1:
            break
        newNode = Node(currData)
        if head is None:
            head = newNode
            tail = newNode
        else:
            tail.next = newNode
            tail = newNode
    return head
def printLL(head):
    while head is not None:
        print(str(head.data)+"->",end="")
        head = head.next
    print("None")
    return
head=takeInput()
printLL(head)
```

```
1 2 3 4 -1
1->2->3->4->None
```

# Quiz

In [18]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def printLL(head):
    while head is not None:
        print(head.data,end=" ")
        head = head.next

def increment(head):
     temp = head
     while temp is not None:
        temp.data +=1
        temp = temp.next



node1 = Node(10)
node2 = Node(20)
node1.next = node2
increment(node1)
printLL(node1)
```

```
11 21
```

# Length of LL

In [19]:

```
"""
Given a linked list, find and return the length of input LL. Do it iteratively.
Linked list elements (separated by space and terminated by -1)
"""
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def length(head):
    ############################
    # PLEASE ADD YOUR CODE HERE #
    ############################
    lis=[]
    while head is not None:
        lis.append(head.data)
        head=head.next
    return len(lis)
def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

# Main
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
len=length(l)
print(len)
```

```
1 2 3 4 5 6 -1
6
```

# Print ith node

In [21]:

```
"""
Given a linked list and a position i, print the node at ith position.
If position i is greater than length of LL, then don't print anything.

Line 1 : Linked list elements (separated by space and terminated by -1)
Line 2 : Integer i (position)
"""
pass
```

In [ ]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def ithNode(head, i):
    ##############################
    # PLEASE ADD YOUR CODE HERE #
    ##############################
    c=0
    while head is not None:
        if c==i:
            return head
        head=head.next
        c+=1
def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

# Main
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
i=int(input())
node = ithNode(l, i)
if node:
    print(node.data)
```

# Insert At Ith Position - Iteratively

In [8]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def takeInput():
    inputList=[int(ele) for ele in input().split()]
    head = None
    tail = None
    for currData in inputList:
        newNode = Node(currData)
        if head is None:
            head = newNode
            tail = newNode
        else:
            tail.next = newNode
            tail = newNode
    return head
def length(head):
    c=0
    current=head
    while current is not None:
        c+=1
        current=current.next
    return c
def insertAtI(data,i,head):
    if i<0 or i>length(head):
        return head
    count = 0
    prev = None
    curr = head
    while count<i:
        prev = curr
        curr = curr.next
        count = count+1

    newNode = Node(data)
    if i==0:
        head = newNode
    else:
        prev.next = newNode

    newNode.next = curr

    return head
def printLL(head):
    while head is not None:
        print(str(head.data)+"->",end="")
        head = head.next
    print("None")
    return
head=takeInput()
printLL(head)
# length(head)
insertAtI(5,2,head)
printLL(head)
insertAtI(6,0,head)
printLL(head)
```

```
1 2 3 4 5
1->2->3->4->5->None
1->2->5->3->4->5->None
1->2->5->3->4->5->None
```

# Delete node

In [1]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def length(head):
    count = 0
    while head is not None:
        count += 1
        head = head.next
    return count-1
def delete(head, i):
    # A linked list and a position i, delete the node of ith position from
    # Linked List iteratively. If position i is greater than length of LL, then
    # you should return the same LL without any change. Indexing starts from 0.
    # You don't need to print the elements, just delete the node and return the
    # head of updated LL.
    ##############################
    # PLEASE ADD YOUR CODE HERE #
    ##############################
    # print(length(head))
    if i<0 or i>length(head):
        return head
    count = 0
    prev=None
    curr=head
    while count<i:
        prev = curr
        curr = curr.next
        count += 1
    if i==0:
        head=curr.next
    else:
        prev.next=curr.next
    return head
def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

def printll(head):
    while head:
        print(head.data, end=' ')
        head = head.next
    print()

# Main
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
i=int(input())
l = delete(l, i)
printll(l)
```

```
1 2 3 4 5 6 -1
2
1 2 4 5 6
```

# Length of LL (recursive)

In [2]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def lengthRecursive(head):
    # A linked list, find and return the length of input LL recursively.
    ############################
    # PLEASE ADD YOUR CODE HERE #
    ############################
    if head is None:
        return 0
    return 1+lengthRecursive(head.next)

def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

# Main
from sys import setrecursionlimit
setrecursionlimit(11000)
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
len=lengthRecursive(l)
print(len)
```

```
1 2 3 4 5 6 -1
6
```

# Insert At Ith Position - Recursively

In [23]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def ll(arr):
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head
def insertAtI(head,data,i):
    if i<0:
        return None
    if head is None:
        return None
    if i==0:
        newNode=Node(data)
        newNode.next=head
        return newNode
    small_ouput=insertAtI(head.next,data,i-1)
    head.next=small_ouput
    return head

def printll(head):
    while head:
        print(str(head.data)+'->',end='')
        head = head.next
    print("None")

# Main
# Read the link list elements including -1
arr=list(map(int,input().split()))
# Create a Linked list after removing -1 from list
head = ll(arr)
printll(head)
insertAtI(head,5,0)
printll(head)
```

```
1 2 3 4
1->2->3->4->None
1->2->3->4->None
```

# Delete node (recursive)

In [ ]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def deleteRec(head, i):
    #  a linked list and a position i, delete the node of ith position from
    # Linked List recursively. If position i is greater than length of LL,
    # then:
    # you should return the same LL without any change.
    ##############################
    # PLEASE ADD YOUR CODE HERE #
    ##############################
    if head is None:
        return
    if i<0:
        return
    if i==0:
        return head.next
    output=deleteRec(head.next,i-1)
    head.next=output
    return head

def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

def printll(head):
    while head:
        print(head.data, end=' ')
        head = head.next
    print()

# Main
from sys import setrecursionlimit
setrecursionlimit(11000)
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
i=int(input())
l = deleteRec(l, i)
printll(l)
```

# Quiz

In [ ]:

```
# What will be the time complexity of searching an element in the linked list ?
# ans=o(n)
```

In [ ]:

```
# Consider the Singly linked list having n elements. What will be the time taken to add
an node at the end of linked list if is initially pointing to first node of the list.
# That is only head is given to you.
#ans=o(n)
```

In [ ]:

```
# There is reference (or pointer) to first Node of the Linked List, then time required
 to insert element to second position is __________.
# Indexing starts from 0.
# ans=o(1)
```

In [ ]:

```
# Given an unsorted singly Linked List, suppose you have reference (or pointer) to its
 head node only, which of the following operation can be implemented in O(1) time?
# i)   Insertion at the front of the linked list
# ii)  Insertion at the end of the linked list
# iii) Deletion of the last node of the linked list
# iv)  Deletion of the front node of the linked list
# ans=i and iv
```

In [ ]:

```
# Given an unsorted singly Linked List, suppose you have references (or pointer) to its
head and tail nodes, which of the following operation can be implemented in O(1) time?
# i)   Insertion at the front of the linked list
# ii)  Insertion at the end of the linked list
# iii) Deletion of the last node of the linked list
# iv)  Deletion of the front node of the linked list
# ans=i,ii and iv
```

# Assignment

# Find a node in LL

In [ ]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def linearSearch(head, n):
    #  Given a linked list and an integer n you need to find and return index
    #  where n is present in the LL. Do this iteratively. Return -1 if n is not
    #  present in the LL. Indexing of nodes starts from 0.
    ##############################
    # PLEASE ADD YOUR CODE HERE #
    ##############################
    c=-1
    while head is not None:
        c+=1
        if head.data==n:
            return c
        head=head.next
    return -1

def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

# Main
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
data=int(input())
index = linearSearch(l, data)
print(index)
```

# AppendLastNToFirst

In [ ]:

```
"""
Given a linked list and an integer n, append the last n elements of the LL to front.
Indexing starts from 0. You don't need to print the elements, just update the elements
 and return the head of updated LL.
Assume given n will be smaller than length of LL.
Input format :

Line 1 : Linked list elements (separated by space and terminated by -1)`

Sample Input 1 :
1 2 3 4 5 -1
3
Sample Output 1 :
3 4 5 1 2
"""
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def length(head):
    c=0
    while head is not None:
        c+=1
        head=head.next
    return c

def append_LinkedList(head,n) :
    #  Given a linked list and an integer n, append the last n elements of the LL
    #  to front.
    ##############################
    # PLEASE ADD YOUR CODE HERE #
    ##############################
    h1=head
    curr=head
    l=length(head)
    i=1
    while i < l-n:
        curr=curr.next
        i+=1
    h2=curr.next
    tail=h2
    curr.next=None
    while True:
        if tail.next is None:
            tail.next=h1
            break
        tail=tail.next
    return h2

def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head
```

```
def printll(head):
    while head:
        print(head.data, end=' ')
        head = head.next
    print()

# Main
# Read the Link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
i=int(input())
l = append_LinkedList(l, i)
printll(l)
```

# Eliminate duplicates from LL

In [1]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def eliminate_duplicate(head):
    #  Given a sorted linked list (elements are sorted in ascending order).
    #  Eliminate duplicates from the given LL, such that output LL contains only
    #  unique elements.
    ##############################
    # PLEASE ADD YOUR CODE HERE #
    ##############################
    h=head
    t=head.next
    while t is not None:
        if h.data!=t.data:
            h.next=t
            t=t.next
            h=h.next
        else:
            t=t.next
    h.next=None
    return head

def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

def printll(head):
    while head:
        print(head.data, end=' ')
        head = head.next
    print()

# Main
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
l = eliminate_duplicate(l)
printll(l)
```

```
1 2 2 2 3 3 3
1 2 3
```

# Print reverse LinkedList

In [2]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def print_linkedlist_spl(head):
    #  Print a given linked list in reverse order. You need to print the tail
    #  first and head last. You can't change any pointer in the linked list, just
    #  print it in reverse order.
    #  GOOD PROBLEM for RECURSION
    ############################
    # PLEASE ADD YOUR CODE HERE #
    ############################
    if head is None:
        return
    print_linkedlist_spl(head.next)
    print(head.data,end=" ")

def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

# Main
# Read the link list elements including -1
from sys import setrecursionlimit
setrecursionlimit(10000)
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
print_linkedlist_spl(l)
```

```
1 2 3 4 5
4 3 2 1
```

# Palindrome LinkedList

In [2]:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def end_node(head):
    while head.next!=None:
        head=head.next
    return head
def prev_node(head,tail):
    while head and head.next!=tail:
        head=head.next
    return head


def check_palindrome(head) :
    ############################
    # PLEASE ADD YOUR CODE HERE #
    ############################
    start=head
    tail=end_node(head)
    while start!=tail and tail.next!=start:
        if start.data!=tail.data:
            return False
        start=start.next
        tail = prev_node(head,tail)
    return True



    # if head.next is None:
    #     return True
    # if head.next.next is None:
    #     v1=head
    #     v2=head.next
    #     if v1.data==v2.data:
    #         return True
    #     else:
    #         return False
    # h=head
    # t=head
    # while t.next is not None and t.next.next is not None:
    #     h=h.next
    #     t=t.next.next
    # f1=head
    # f2=h.next
    # h.next=None
    # r=rev(f2)
    # while f1 is not None and f2 is not None:
    #     if f1.data!=f2.data:
    #         return False
    #     f1=f1.next
    #     f2=f2.next
    # return True


    # while f1 is not None:
    #     print(f1.data,end=" ")
    #     f1=f1.next
    # print()
```

```
    # while r is not None:
    #     print(r.data,end=" ")
    #     r=r.next



def ll(arr):
    if len(arr)==0:
        return None
    head = Node(arr[0])
    last = head
    for data in arr[1:]:
        last.next = Node(data)
        last = last.next
    return head

# Main
# Read the link list elements including -1
arr=list(int(i) for i in input().strip().split(' '))
# Create a Linked list after removing -1 from list
l = ll(arr[:-1])
ans = check_palindrome(l)
if ans:
    print("true")
else:
    print("false")
```

```
1 2 2 1 -1
true
```

# ends of assignment

In [ ]: