In [1]:

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

a = Node(13)
b = Node(15)
a.next = b
print(a.data)
print(b.data)
print(a.next.data)
```

```
13
15
15
```

In [2]:

```
print(a)
print(a.next)
print(b)
```

```
<__main__.Node object at 0x00000255BF074610>
<__main__.Node object at 0x00000255BF074670>
<__main__.Node object at 0x00000255BF074670>
```

In [3]:

```
print(b.next.data)
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp/ipykernel_15028/3284088142.py in <module>
----> 1 print(b.next.data)

AttributeError: 'NoneType' object has no attribute 'data'
```

In [12]:

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

a = Node(2)
b = Node(3)
print(a.data)
print(b.data)
print(a.next)
a.next = b
print(b)
print(a.next)
b.next = a
print(b.next)
```

```
2
3
None
<__main__.Node object at 0x00000255BF074C40>
<__main__.Node object at 0x00000255BF074C40>
<__main__.Node object at 0x00000255BF074850>
```

In [16]:

```
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 [17]:

```
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
```

In [21]:

```
type(print(node4.next))
```

```
None
```

Out[21]:

```
NoneType
```

Taking inputs of linked list

In [7]:

```
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

head = takeInput()
printLL(head)
```

```
7 8 9 6 5 -1
7->8->9->6->5->None
```

In [6]:

```
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

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

head = takeInput()
printLL(head)
```

```
5 6 9 3 8 -1
5->6->9->3->8->None
```

Printing linked list

In [8]:

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

def printLL(head):

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

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

head = takeInput()
printLL(head)
```

```
7 8 9 6 5 -1
7->8->9->6->5->None
```

OptimizedInput

In [5]:

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

def printLL(head):

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

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

head = takeInput()
#a = printLL(head)
length(head)
```

```
7 8 9 6 2 -1
```

Out[5]:

```
5
```

In [ ]:

```
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)
```

Getting length of linked list

In [5]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count




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

head = takeInput()
#a = printLL(head)
length(head)

```

```
7 8 9 6 42 -1
```

Out[5]:

```
5
```

Print ith index element

In [1]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count

def print_ith(head,i):
    if i<0 or i>length(head):
        return None

    count = 0
    curr = head
    while count<i:
        curr = curr.next
        count = count + 1
    r = curr.data
    return r

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

head = takeInput()
printLL(head)
print_ith(head,3)
```

```
4 5 6 3 2 5 8 1 2 3 6 9 -1
4->5->6->3->2->5->8->1->2->3->6->9->None
```

Out[1]:

3

Inserting at ith position

In [8]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next

    return count


def insert_ith(head,i,data):
    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 prev is not None:
        prev.next = newNode
    else:
        head = newNode
    newNode.next = curr

    return head


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


head = takeInput()
printLL(head)
"""head = insert_ith(head,2,5)
printLL(head)"""
head = insert_ith(head,0,9)
printLL(head)
```

```
4 5 6 3 2 5 -1
4->5->6->3->2->5->None
9->4->5->6->3->2->5->None
```

Deleting ith node

In [6]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next

    return count


def insert_ith(head,i,data):
    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 prev is not None:
        prev.next = newNode
    else:
        head = newNode
    newNode.next = curr

    return head

def delete_ith(head,i):
    if i<0 or i>length(head):
        return None

    count = 0
    prev = None
    curr = head
    while count<i:
        prev=curr
        curr=curr.next
        count = count+1

    if prev is not None:
        prev.next = curr.next
    else:
        head = curr.next
```

```
    del curr

    return head


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

head = takeInput()
printLL(head)
head = delete_ith(head,0)
printLL(head)
```

```
7 8 9 6 2 -1
7->8->9->6->2->None
8->9->6->2->None
```

Inserting at ith index Recursively

In [15]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next

    return count


def insert_ith(head,i,data):
    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 prev is not None:
        prev.next = newNode
    else:
        head = newNode
    newNode.next = curr

    return head

def insert_i_recur(head,i,data):
    if i<0:
        return head

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

    if head is None:
        return None

    smallHead = insert_i_recur(head.next, i-1, data)
    head.next = smallHead
    return head

```

```python
def delete_ith(head,i):
    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

    if prev is not None:
        prev.next = curr.next
    else:
        head = curr.next
    del curr

    return head

def delete_i_recur(head,i):

    if i==0:
        curr = None
        curr = head.next
        del head
        return curr
    if head is None:
        return None

    smallHead = delete_i_recur(head.next, i-1)
    head.next = smallHead
    return head




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

head = takeInput()
printLL(head)
head = insert_i_recur(head,2,4)
printLL(head)
```

```
7 8 9 6 3 25 -1
7->8->9->6->3->25->None
7->8->4->9->6->3->25->None
```

Delete the ith element recursively

In [ ]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next

    return count


def insert_ith(head,i,data):
    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 prev is not None:
        prev.next = newNode
    else:
        head = newNode
    newNode.next = curr

    return head

def insert_i_recur(head,i,data):
    if i<0:
        return head

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

    if head is None:
        return None

    smallHead = insert_i_recur(head.next, i-1, data)
    head.next = smallHead
    return head

```

```python
def delete_ith(head,i):
    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

    if prev is not None:
        prev.next = curr.next
    else:
        head = curr.next
    del curr

    return head

def delete_i_recur(head,i):

    if i==0:
        curr = None
        curr = head.next
        del head
        return curr
    if head is None:
        return None

    smallHead = delete_i_recur(head.next, i-1)
    head.next = smallHead
    return head




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

head = takeInput()
printLL(head)
head = delete_i_recur(head,2)
printLL(head)
head = delete_i_recur(head,1)
```

```
121 printLL(head)
```

Searching for a node and returning its index

In [4]:

```
 1 class Node:
 2     def __init__(self,data):
 3         self.data = data
 4         self.next = None
 5 
 6 def printLL(head):
 7 
 8     while head is not None:
 9         print(str(head.data)+"->",end="")
10         head = head.next
11     print("None")
12     return
13 
14 def search_node(head,data):
15 
16     curr = head
17     count = 0
18 
19     while curr.data != data:
20         count = count+1
21         curr = curr.next
22     return count
23 
24 def takeInput():
25 
26     inputList = [int(ele) for ele in input().split()]
27     head = None
28     tail = None
29     for currData in inputList:
30         if currData == -1:
31             break
32 
33         newNode = Node(currData)
34         if head is None:
35             head = newNode
36             tail = newNode
37         else:
38             tail.next = newNode
39             tail = newNode
40     return head
41 
42 head = takeInput()
43 printLL(head)
44 search_node(head,8)
```

```
7 8 5 2 3 -1
7->8->5->2->3->None
```

Out[4]:

```
1
```

Append last N elements in first of linked list

In [11]:

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

def printLL(head):

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

def search_node(head,data):

    curr = head
    count = 0

    while curr.data != data:
        count = count+1
        curr = curr.next
    return count

def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next

    return count

def append_last_n_node(head,n):

    if n == 0:
        return head
    if n == length(head):
        return head
    count = 1
    curr = head
    while count<length(head)-n:
        curr = curr.next
        count = count + 1

    head2 = curr.next
    curr.next = None
    curr2 = head2
    while curr2.next is not None:
        curr2 = curr2.next
    curr2.next = head

    return head2

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

head = takeInput()
printLL(head)
head = append_last_n_node(head,0)
printLL(head)
```

```
7 8 5 6 3 -1
7->8->5->6->3->None
7->8->5->6->3->None
```

Removing duplicated values from the linked list

In [19]:

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

def printLL(head):

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

def search_node(head,data):

    curr = head
    count = 0

    while curr.data != data:
        count = count+1
        curr = curr.next
    return count

def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next

    return count

def append_last_n_node(head,n):

    if n == 0:
        return head
    if n == length(head):
        return head
    count = 1
    curr = head
    while count<length(head)-n:
        curr = curr.next
        count = count + 1

    head2 = curr.next
    curr.next = None
    curr2 = head2
    while curr2.next is not None:
        curr2 = curr2.next
    curr2.next = head

    return head2


def eliminate_duplicates(head):
    if head is None or head.next is None:
        return head

    current_node = head
    while current_node is not None:
```

```
        runner = current_node
        while runner.next is not None:
            if runner.next.data == current_node.data:
                runner.next = runner.next.next
            else:
                runner = runner.next

        current_node = current_node.next

    return head

"""def eliminate_duplicate(head):
    t1 = head
    t2 = t1.next

    if head is None or head.next == None:
        return head


        while t2 is not None:
            if t1 != t2:
                t1.next = t2
                t1 = t2
                t2 = t2.next

            elif t1 == t2:
                t2 = t2.next

        t1.next = None
        return head
"""


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

head = takeInput()
printLL(head)
head = eliminate_duplicates(head)
printLL(head)
```

```
7 8 8 9 9 6 6 -1
7->8->8->9->9->6->6->None
7->8->9->6->None
```

Reversing a linked list using Recursion

In [22]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count


def reversel(head):

    if head is None or head.next is None:
        return head

    smallhead = reversel(head.next)
    curr = smallhead
    while curr.next is not None:
        curr = curr.next
    curr.next = head
    head.next = None
    return smallhead

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

head = takeInput()
printLL(head)
head = reversel(head)
```

```
58 printLL(head)
```

```
4 5 26 8 5 -1
4->5->26->8->5->None
5->8->26->5->4->None
```

Reversing a linked list

In [3]:

```
class Node:

    def __init__(self):
        self.data = data
        self.next = None


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



def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count


def reverse_LL(head):
    count = 1
    n = length(head)
    curr = head
    while head is not None:
        count = count +  1
        curr = curr.next
        if count == n-1:
            print(curr.data, end=" ")
            n = n-1

    return head

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

head = takeInput()
printLL(head)
a = reverse_LL(head)
```

```
60 print(a)
```

```
7 8 5 2 3 -1

---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp/ipykernel_19644/1513982713.py in <module>
     55     return head
     56
---> 57 head = takeInput()
     58 printLL(head)
     59 a = reverse_LL(head)

~\AppData\Local\Temp/ipykernel_19644/1513982713.py in takeInput()
     45             break
     46
---> 47         newNode = Node(currData)
     48         if head is None:
     49             head = newNode

TypeError: __init__() takes 1 positional argument but 2 were given
```

Finding mid point of Linked list

In [12]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count


def middle(head):
        slow=fast=head
        if(fast is not None):
            while(fast is not None and fast.next.next is not None):
                slow=slow.next
                fast=fast.next.next
            print("Middle is: ",slow.data)

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

head = takeInput()
printLL(head)
middle(head)

```

```
7 8 5 6 9 2 -1
7->8->5->6->9->2->None
Middle is:  5
```

Merge two sorted linked list

In [13]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count

def mid_point_LL(head):

    count = 1
    curr = head
    while count<(length(head)-1)/2:
        curr = curr.next
        count = count + 1
    a = curr.data
    return a

def merge_sorted_LL(head1, head2):

    fh = None
    ft = None
    if (head1 is None) and (head2 is None):
        return head1
    if (head1 is not None) and (head2 is None):
        return head1
    if (head1 is None) and (head2 is not None):
        return head2
    if head1.data < head2.data:
        fh = head1
        ft = head1
        head1 = head1.next
    else:
        fh = head2
        ft = head2
        head2 = head2.next

    while (head1 != None) and (head2 != None):

        if head1.data < head2.data:
            ft.next = head1
            ft = ft.next
            head1 = head1.next

        else:
            ft.next = head2
```

```
            ft = ft.next
            head2 = head2.next

    if head1!=None:
        ft.next = head1
    if head2!=None:
        ft.next = head2

    return fh

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

head1 = takeInput()
printLL(head1)
head2 = takeInput()
printLL(head2)
head = merge_sorted_LL(head1,head2)
printLL(head)
```

```
1 2 6 7 -1
1->2->6->7->None
4 5 8 9 -1
4->5->8->9->None
1->2->4->5->6->7->8->9->None
```

In [4]:

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

def printLL(head):

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


def length(head):
    count = 0
    while head is not None:
        count = count + 1
        head = head.next
    return count

def mid_point_LL(head):

    count = 1
    curr = head
    while count<(length(head)-1)/2:
        curr = curr.next
        count = count + 1
    a = curr.data
    return a

def merge_sorted_LL(head1, head2):

    fh = None
    ft = None
    if (head1 is None) and (head2 is None):
        return head1
    if (head1 is not None) and (head2 is None):
        return head1
    if (head1 is None) and (head2 is not None):
        return head2
    if head1.data < head2.data:
        fh = head1
        ft = head1
        head1 = head1.next
    else:
        fh = head2
        ft = head2
        head2 = head2.next

    while (head1 != None) and (head2 != None):

        if head1.data < head2.data:
            ft.next = head1
            ft = ft.next
            head1 = head1.next

        else:
            ft.next = head2
```

```
            ft = ft.next
            head2 = head2.next

    if head1!=None:
        ft.next = head1
    if head2!=None:
        ft.next = head2

    return fh

def reverseLL(head):
    if head == None:
        return None
    elif head != None and head.next == None:
        return head
    else:
        temp = None
        next_node = None
        while head != None:
            next_node = head.next
            head.next = temp
            temp = head
            head = next_node
        return temp

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

head = takeInput()
printLL(head)
head = reverseLL(head)
print(head)
```

```
8 5 2 3 66 -1
8->5->2->3->66->None
<__main__.Node object at 0x00000191AF0792B0>
```

In [14]:

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

def printLL(head):

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


def Even_odd_LL(head):
    if head is None:
        return None

    if head.next is None:
        return head

    e_h, e_t = None, None
    o_h, o_t = None, None

    while head!=None:

        if head.data %2 != 0:
            o_h, o_t = head, head
            head = head.next

        if head.data %2 == 0:
            e_h, e_t = head, head
            head = head.next




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

head1 = takeInput()
printLL(head)
head = Even_odd(head)
```

```
60 printLL(head)
```

```
7 5 3 6 2 4 58 9 6 3 2 1 -1
1->2->4->5->6->7->8->9->None

---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp/ipykernel_19644/3744214114.py in <module>
     88 head1 = takeInput()
     89 printLL(head)
---> 90 head = Even_odd(head)
     91 printLL(head)

~\AppData\Local\Temp/ipykernel_19644/3744214114.py in Even_odd(head)
     58             even_t = head
     59
---> 60         even_t.next = head
     61         even_t = even_t.next
     62

AttributeError: 'NoneType' object has no attribute 'next'
```

In [7]:

```
1 a = 4
2
```

In [ ]:

```
1
```