

In [1]:

```
1 class Node:
2     def __init__(self,data):
3         self.data = data
4         self.next = None
5
6 a = Node(13)
7 b = Node(15)
8 a.next = b
9 print(a.data)
10 print(b.data)
11 print(a.next.data)
```

```
13
15
15
```

In [2]:

```
1 print(a)
2 print(a.next)
3 print(b)
```

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

In [3]:

```
1 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]:

```
1 class Node:
2     def __init__(self,data):
3         self.data = data
4         self.next = None
5
6 a = Node(2)
7 b = Node(3)
8 print(a.data)
9 print(b.data)
10 print(a.next)
11 a.next = b
12 print(b)
13 print(a.next)
14 b.next = a
15 print(b.next)
```

2
3

None

```
<__main__.Node object at 0x00000255BF074C40>
<__main__.Node object at 0x00000255BF074C40>
<__main__.Node object at 0x00000255BF074850>
```

In [16]:

```
1 class Node:
2     def __init__(self,data):
3         self.data = data
4         self.next = None
5
6 def printLL(head):
7     while head is not None:
8         print(head.data, end=" ")
9         head = head.next
10
11 node1 = Node(10)
12 node2 = Node(20)
13 node2.next = node1
14 printLL(node2)
```

20 10

In [17]:

```
1 class Node:
2     def __init__(self,data):
3         self.data = data
4         self.next = None
5
6 def printLL(head):
7     while head is not None:
8         print(head.data, end=" ")
9         head = head.next
10
11 node1 = Node(10)
12 node2 = Node(20)
13 node3 = Node(30)
14 node4 = Node(40)
15 node1.next = node2
16 node2.next = node3
17 node3.next = node4
18 printLL(node2)
```

20 30 40

In [21]:

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

None

Out[21]:

NoneType

Taking inputs of linked list

In [7]:

```
1 class Node:
2     def __init__(self,data):
3         self.data = data
4         self.next = None
5
6 def takeInput():
7
8     inputList = [int(ele) for ele in input().split()]
9     head = None
10    for currData in inputList:
11        if currData == -1:
12            break
13
14        newNode = Node(currData)
15        if head is None:
16            head = newNode
17        else:
18            curr = head
19            while curr.next is not None:
20                curr = curr.next
21            curr.next = newNode
22    return head
23
24 head = takeInput()
25 printLL(head)
```

7 8 9 6 5 -1

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

In [6]:

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

5 6 9 3 8 -1

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

Printing linked list

In [8]:

```
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 takeInput():
15
16    inputList = [int(ele) for ele in input().split()]
17    head = None
18    for currData in inputList:
19        if currData == -1:
20            break
21
22        newNode = Node(currData)
23        if head is None:
24            head = newNode
25        else:
26            curr = head
27            while curr.next is not None:
28                curr = curr.next
29            curr.next = newNode
30    return head
31
32 head = takeInput()
33 printLL(head)
```

7 8 9 6 5 -1

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

OptimizedInput

In [5]:

```
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 takeInput():
15
16     inputList = [int(ele) for ele in input().split()]
17     head = None
18     tail = None
19     for currData in inputList:
20         if currData == -1:
21             break
22
23         newNode = Node(currData)
24         if head is None:
25             head = newNode
26             tail = newNode
27         else:
28             tail.next = newNode
29             tail = newNode
30
31     return head
32
33 head = takeInput()
34 #a = printLL(head)
35 length(head)
```

7 8 9 6 2 -1

Out[5]:

5

In []:

```
1 class Node:
2     def __init__(self,data):
3         self.data = data
4         self.next = None
5
6 def printLL(head):
7     while head is not None:
8         print(head.data, end="")
9         head=head.next
10
11 def increment(head):
12     temp = head
13     while temp is not None:
14         temp.data+=1
15         temp=temp.next
16
17 node1 = Node(10)
18 node2 = Node(20)
19 node1.next = node2
20 increment(node1)
21 printLL(node1)
```

Getting length of linked list

In [5]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20     return count
21
22
23
24
25 def takeInput():
26
27     inputList = [int(ele) for ele in input().split()]
28     head = None
29     tail = None
30     for currData in inputList:
31         if currData == -1:
32             break
33
34         newNode = Node(currData)
35         if head is None:
36             head = newNode
37             tail = newNode
38         else:
39             tail.next = newNode
40             tail = newNode
41
42     return head
43
44 head = takeInput()
45 #a = printLL(head)
46 length(head)
47
```

7 8 9 6 42 -1

Out[5]:

5

Print ith index element

In [1]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20     return count
21
22 def print_ith(head,i):
23     if i<0 or i>length(head):
24         return None
25
26     count = 0
27     curr = head
28     while count<i:
29         curr = curr.next
30         count = count + 1
31     r = curr.data
32     return r
33
34 def takeInput():
35
36     inputList = [int(ele) for ele in input().split()]
37     head = None
38     tail = None
39     for currData in inputList:
40         if currData == -1:
41             break
42
43         newNode = Node(currData)
44         if head is None:
45             head = newNode
46             tail = newNode
47         else:
48             tail.next = newNode
49             tail = newNode
50
51     return head
52
53 head = takeInput()
54 printLL(head)
55 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]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20
21     return count
22
23
24 def insert_ith(head,i,data):
25     if i<0 or i>length(head):
26         return head
27
28     count = 0
29     prev = None
30     curr = head
31     while count<i:
32         prev = curr
33         curr = curr.next
34         count = count + 1
35     newNode = Node(data)
36     if prev is not None:
37         prev.next = newNode
38     else:
39         head = newNode
40     newNode.next = curr
41
42     return head
43
44
45 def takeInput():
46
47     inputList = [int(ele) for ele in input().split()]
48     head = None
49     tail = None
50     for currData in inputList:
51         if currData == -1:
52             break
53
54         newNode = Node(currData)
55         if head is None:
56             head = newNode
57             tail = newNode
58         else:
59             tail.next = newNode
```

```
60         tail = newNode
61
62     return head
63
64
65 head = takeInput()
66 printLL(head)
67 """head = insert_ith(head,2,5)
68 printLL(head)"""
69 head = insert_ith(head,0,9)
70 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]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20
21     return count
22
23
24 def insert_ith(head,i,data):
25     if i<0 or i>length(head):
26         return head
27
28     count = 0
29     prev = None
30     curr = head
31     while count<i:
32         prev = curr
33         curr = curr.next
34         count = count + 1
35     newNode = Node(data)
36     if prev is not None:
37         prev.next = newNode
38     else:
39         head = newNode
40     newNode.next = curr
41
42     return head
43
44 def delete_ith(head,i):
45     if i<0 or i>length(head):
46         return None
47
48     count = 0
49     prev = None
50     curr = head
51     while count<i:
52         prev=curr
53         curr=curr.next
54         count = count+1
55
56     if prev is not None:
57         prev.next = curr.next
58     else:
59         head = curr.next
```



```

60     del curr
61
62     return head
63
64
65 def takeInput():
66
67     inputList = [int(ele) for ele in input().split()]
68     head = None
69     tail = None
70     for currData in inputList:
71         if currData == -1:
72             break
73
74         newNode = Node(currData)
75         if head is None:
76             head = newNode
77             tail = newNode
78         else:
79             tail.next = newNode
80             tail = newNode
81
82     return head
83
84 head = takeInput()
85 printLL(head)
86 head = delete_ith(head,0)
87 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]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20
21     return count
22
23
24 def insert_ith(head,i,data):
25     if i<0 or i>length(head):
26         return head
27
28     count = 0
29     prev = None
30     curr = head
31     while count<i:
32         prev = curr
33         curr = curr.next
34         count = count + 1
35     newNode = Node(data)
36     if prev is not None:
37         prev.next = newNode
38     else:
39         head = newNode
40     newNode.next = curr
41
42     return head
43
44 def insert_i_recur(head,i,data):
45     if i<0:
46         return head
47
48     if i==0:
49         newNode = Node(data)
50         newNode.next = head
51         return newNode
52
53     if head is None:
54         return None
55
56     smallHead = insert_i_recur(head.next, i-1, data)
57     head.next = smallHead
58     return head
59
```

```

60 def delete_ith(head,i):
61     if i<0 or i>length(head):
62         return head
63
64     count = 0
65     prev = None
66     curr = head
67     while count<i:
68         prev=curr
69         curr=curr.next
70         count = count+1
71
72     if prev is not None:
73         prev.next = curr.next
74     else:
75         head = curr.next
76     del curr
77
78     return head
79
80 def delete_i_recur(head,i):
81
82     if i==0:
83         curr = None
84         curr = head.next
85         del head
86         return curr
87     if head is None:
88         return None
89
90     smallHead = delete_i_recur(head.next, i-1)
91     head.next = smallHead
92     return head
93
94
95
96
97 def takeInput():
98
99     inputList = [int(ele) for ele in input().split()]
100     head = None
101     tail = None
102     for currData in inputList:
103         if currData == -1:
104             break
105
106         newNode = Node(currData)
107         if head is None:
108             head = newNode
109             tail = newNode
110         else:
111             tail.next = newNode
112             tail = newNode
113
114     return head
115
116 head = takeInput()
117 printLL(head)
118 head = insert_i_recur(head,2,4)
119 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 []:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20
21     return count
22
23
24 def insert_ith(head,i,data):
25     if i<0 or i>length(head):
26         return head
27
28     count = 0
29     prev = None
30     curr = head
31     while count<i:
32         prev = curr
33         curr = curr.next
34         count = count + 1
35     newNode = Node(data)
36     if prev is not None:
37         prev.next = newNode
38     else:
39         head = newNode
40     newNode.next = curr
41
42     return head
43
44 def insert_i_recur(head,i,data):
45     if i<0:
46         return head
47
48     if i==0:
49         newNode = Node(data)
50         newNode.next = head
51         return newNode
52
53     if head is None:
54         return None
55
56     smallHead = insert_i_recur(head.next, i-1, data)
57     head.next = smallHead
58     return head
59
```

```

60 def delete_ith(head,i):
61     if i<0 or i>length(head):
62         return head
63
64     count = 0
65     prev = None
66     curr = head
67     while count<i:
68         prev=curr
69         curr=curr.next
70         count = count+1
71
72     if prev is not None:
73         prev.next = curr.next
74     else:
75         head = curr.next
76     del curr
77
78     return head
79
80 def delete_i_recur(head,i):
81
82     if i==0:
83         curr = None
84         curr = head.next
85         del head
86         return curr
87     if head is None:
88         return None
89
90     smallHead = delete_i_recur(head.next, i-1)
91     head.next = smallHead
92     return head
93
94
95
96
97 def takeInput():
98
99     inputList = [int(ele) for ele in input().split()]
100     head = None
101     tail = None
102     for currData in inputList:
103         if currData == -1:
104             break
105
106         newNode = Node(currData)
107         if head is None:
108             head = newNode
109             tail = newNode
110         else:
111             tail.next = newNode
112             tail = newNode
113
114     return head
115
116 head = takeInput()
117 printLL(head)
118 head = delete_i_recur(head,2)
119 printLL(head)
120 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]:

```
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 length(head):
25     count = 0
26     while head is not None:
27         count = count + 1
28         head = head.next
29
30     return count
31
32 def append_last_n_node(head,n):
33
34     if n == 0:
35         return head
36     if n == length(head):
37         return head
38     count = 1
39     curr = head
40     while count<length(head)-n:
41         curr = curr.next
42         count = count + 1
43
44     head2 = curr.next
45     curr.next = None
46     curr2 = head2
47     while curr2.next is not None:
48         curr2 = curr2.next
49     curr2.next = head
50
51     return head2
52
53 def takeInput():
54
55     inputList = [int(ele) for ele in input().split()]
56     head = None
57     tail = None
58     for currData in inputList:
59         if currData == -1:
```

```
60         break
61
62     newNode = Node(currData)
63     if head is None:
64         head = newNode
65         tail = newNode
66     else:
67         tail.next = newNode
68         tail = newNode
69     return head
70
71 head = takeInput()
72 printLL(head)
73 head = append_last_n_node(head,0)
74 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]:

```
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 length(head):
25     count = 0
26     while head is not None:
27         count = count + 1
28         head = head.next
29
30     return count
31
32 def append_last_n_node(head,n):
33
34     if n == 0:
35         return head
36     if n == length(head):
37         return head
38     count = 1
39     curr = head
40     while count<length(head)-n:
41         curr = curr.next
42         count = count + 1
43
44     head2 = curr.next
45     curr.next = None
46     curr2 = head2
47     while curr2.next is not None:
48         curr2 = curr2.next
49     curr2.next = head
50
51     return head2
52
53
54 def eliminate_duplicates(head):
55     if head is None or head.next is None:
56         return head
57
58     current_node = head
59     while current_node is not None:
```

```

60     runner = current_node
61     while runner.next is not None:
62         if runner.next.data == current_node.data:
63             runner.next = runner.next.next
64         else:
65             runner = runner.next
66
67     current_node = current_node.next
68
69     return head
70
71 """def eliminate_duplicate(head):
72     t1 = head
73     t2 = t1.next
74
75     if head is None or head.next == None:
76         return head
77
78
79     while t2 is not None:
80         if t1 != t2:
81             t1.next = t2
82             t1 = t2
83             t2 = t2.next
84
85         elif t1 == t2:
86             t2 = t2.next
87
88     t1.next = None
89     return head
90 """
91
92
93 def takeInput():
94
95     inputList = [int(ele) for ele in input().split()]
96     head = None
97     tail = None
98     for currData in inputList:
99         if currData == -1:
100             break
101
102         newNode = Node(currData)
103         if head is None:
104             head = newNode
105             tail = newNode
106         else:
107             tail.next = newNode
108             tail = newNode
109     return head
110
111 head = takeInput()
112 printLL(head)
113 head = eliminate_duplicates(head)
114 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]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20     return count
21
22
23 def reversel(head):
24
25     if head is None or head.next is None:
26         return head
27
28     smallhead = reversel(head.next)
29     curr = smallhead
30     while curr.next is not None:
31         curr = curr.next
32     curr.next = head
33     head.next = None
34     return smallhead
35
36 def takeInput():
37
38     inputList = [int(ele) for ele in input().split()]
39     head = None
40     tail = None
41     for currData in inputList:
42         if currData == -1:
43             break
44
45         newNode = Node(currData)
46         if head is None:
47             head = newNode
48             tail = newNode
49         else:
50             tail.next = newNode
51             tail = newNode
52
53     return head
54
55 head = takeInput()
56 printLL(head)
57 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]:

```
1 class Node:
2
3     def __init__(self):
4         self.data = data
5         self.next = None
6
7
8 def printLL(head):
9     while head is not None:
10         print(str(head.data)+"->",end="")
11         head = head.next
12     print("None")
13     return
14
15
16
17 def length(head):
18     count = 0
19     while head is not None:
20         count = count + 1
21         head = head.next
22     return count
23
24
25 def reverse_LL(head):
26     count = 1
27     n = length(head)
28     curr = head
29     while head is not None:
30         count = count + 1
31         curr = curr.next
32         if count == n-1:
33             print(curr.data, end=" ")
34             n = n-1
35
36     return head
37
38 def takeInput():
39
40     inputList = [int(ele) for ele in input().split()]
41     head = None
42     tail = None
43     for currData in inputList:
44         if currData == -1:
45             break
46
47         newNode = Node(currData)
48         if head is None:
49             head = newNode
50             tail = newNode
51         else:
52             tail.next = newNode
53             tail = newNode
54
55     return head
56
57 head = takeInput()
58 printLL(head)
59 a = reverse_LL(head)
```

```
60 print(a)
```

```
7 8 5 2 3 -1
```

```
-----  
-  
TypeError                                Traceback (most recent call last)
```

```
~\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]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20     return count
21
22
23 def middle(head):
24     slow=fast=head
25     if(fast is not None):
26         while(fast is not None and fast.next.next is not None):
27             slow=slow.next
28             fast=fast.next.next
29     print("Middle is: ",slow.data)
30
31 def takeInput():
32
33     inputList = [int(ele) for ele in input().split()]
34     head = None
35     tail = None
36     for currData in inputList:
37         if currData == -1:
38             break
39
40         newNode = Node(currData)
41         if head is None:
42             head = newNode
43             tail = newNode
44         else:
45             tail.next = newNode
46             tail = newNode
47
48     return head
49
50 head = takeInput()
51 printLL(head)
52 middle(head)
53
```

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

Merge two sorted linked list

In [13]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20     return count
21
22 def mid_point_LL(head):
23
24     count = 1
25     curr = head
26     while count<(length(head)-1)/2:
27         curr = curr.next
28         count = count + 1
29     a = curr.data
30     return a
31
32 def merge_sorted_LL(head1, head2):
33
34     fh = None
35     ft = None
36     if (head1 is None) and (head2 is None):
37         return head1
38     if (head1 is not None) and (head2 is None):
39         return head1
40     if (head1 is None) and (head2 is not None):
41         return head2
42     if head1.data < head2.data:
43         fh = head1
44         ft = head1
45         head1 = head1.next
46     else:
47         fh = head2
48         ft = head2
49         head2 = head2.next
50
51     while (head1 != None) and (head2 != None):
52
53         if head1.data < head2.data:
54             ft.next = head1
55             ft = ft.next
56             head1 = head1.next
57
58         else:
59             ft.next = head2
```

```

60         ft = ft.next
61         head2 = head2.next
62
63     if head1!=None:
64         ft.next = head1
65     if head2!=None:
66         ft.next = head2
67
68     return fh
69
70 def takeInput():
71
72     inputList = [int(ele) for ele in input().split()]
73     head = None
74     tail = None
75     for currData in inputList:
76         if currData == -1:
77             break
78
79         newNode = Node(currData)
80         if head is None:
81             head = newNode
82             tail = newNode
83         else:
84             tail.next = newNode
85             tail = newNode
86
87     return head
88
89 head1 = takeInput()
90 printLL(head1)
91 head2 = takeInput()
92 printLL(head2)
93 head = merge_sorted_LL(head1,head2)
94 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]:

```
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
15 def length(head):
16     count = 0
17     while head is not None:
18         count = count + 1
19         head = head.next
20     return count
21
22 def mid_point_LL(head):
23
24     count = 1
25     curr = head
26     while count<(length(head)-1)/2:
27         curr = curr.next
28         count = count + 1
29     a = curr.data
30     return a
31
32 def merge_sorted_LL(head1, head2):
33
34     fh = None
35     ft = None
36     if (head1 is None) and (head2 is None):
37         return head1
38     if (head1 is not None) and (head2 is None):
39         return head1
40     if (head1 is None) and (head2 is not None):
41         return head2
42     if head1.data < head2.data:
43         fh = head1
44         ft = head1
45         head1 = head1.next
46     else:
47         fh = head2
48         ft = head2
49         head2 = head2.next
50
51     while (head1 != None) and (head2 != None):
52
53         if head1.data < head2.data:
54             ft.next = head1
55             ft = ft.next
56             head1 = head1.next
57
58         else:
59             ft.next = head2
```

```

60         ft = ft.next
61         head2 = head2.next
62
63     if head1!=None:
64         ft.next = head1
65     if head2!=None:
66         ft.next = head2
67
68     return fh
69
70 def reverseLL(head):
71     if head == None:
72         return None
73     elif head != None and head.next == None:
74         return head
75     else:
76         temp = None
77         next_node = None
78         while head != None:
79             next_node = head.next
80             head.next = temp
81             temp = head
82             head = next_node
83         return temp
84
85 def takeInput():
86
87     inputList = [int(ele) for ele in input().split()]
88     head = None
89     tail = None
90     for currData in inputList:
91         if currData == -1:
92             break
93
94         newNode = Node(currData)
95         if head is None:
96             head = newNode
97             tail = newNode
98         else:
99             tail.next = newNode
100             tail = newNode
101
102     return head
103
104 head = takeInput()
105 printLL(head)
106 head = reverseLL(head)
107 print(head)

```

8 5 2 3 66 -1

8->5->2->3->66->None

<__main__.Node object at 0x00000191AF0792B0>

In [14]:

```
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
15 def Even_odd_LL(head):
16     if head is None:
17         return None
18
19     if head.next is None:
20         return head
21
22     e_h, e_t = None, None
23     o_h, o_t = None, None
24
25     while head!=None:
26
27         if head.data %2 != 0:
28             o_h, o_t = head, head
29             head = head.next
30
31         if head.data %2 == 0:
32             e_h, e_t = head, head
33             head = head.next
34
35
36
37
38 def takeInput():
39
40     inputList = [int(ele) for ele in input().split()]
41     head = None
42     tail = None
43     for currData in inputList:
44         if currData == -1:
45             break
46
47         newNode = Node(currData)
48         if head is None:
49             head = newNode
50             tail = newNode
51         else:
52             tail.next = newNode
53             tail = newNode
54
55     return head
56
57 head1 = takeInput()
58 printLL(head1)
59 head = Even_odd(head1)
```



```
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
```