

**Given a singly linked list, find the middle of the linked list.**

In [18]:

```
1 class Node:
2     def __init__(self,data):
3         self.data=data
4         self.next=None
5 class CreateLinkedList:
6     def __init__(self):
7         self.head=None
8
9     def addElement(self,data):
10        new_data= Node(data)
11
12        if(self.head==None):
13            self.head=new_data
14            return
15        temp=self.head
16        while(temp.next!=None):
17            temp=temp.next
18        temp.next=new_data
19    def length(self):
20        count=1
21        temp=self.head
22
23        while(temp.next!=None):
24            count+=1
25            temp=temp.next
26
27        return count
28    def middleElement(self):
29        len1=list1.length()
30        even=0
31        odd=0
32        print("Length", len1)
33        if(len1%2==0):
34            even=len1//2+1
35            return list1.middle(even)
36        else:
37            odd=len1//2+1
38            return list1.middle(odd)
39    def middle(self,n):
```

```
40         temp=self.head
41         for i in range(n-1):
42             temp=temp.next
43         return temp.data
44
45
46     def print1(self):
47         temp=self.head
48         while(temp):
49             print(temp.data,end=" ")
50             temp=temp.next
51         print()
52 list1=CreateLinkList()
53 list1.addElement(1)
54 list1.addElement(2)
55 list1.addElement(3)
56 list1.addElement(4)
57 list1.addElement(5)
58 print("Created Linked List",end=" ")
59 list1.print1()
60 print("Middle Element",list1.middleElement())
61
```

Created Linked List 1 2 3 4 5

Length 5

Middle Element 3

**Given a singly linked list consisting of N nodes. The task is to remove duplicates (nodes with duplicate values) from the given list (if exists).**

Note: Try not to use extra space. Expected time complexity is  $O(N)$ . The nodes are arranged in a sorted way

In [13]:

```
1 class Node:
2     def __init__(self,data):
3         self.data=data
4         self.next=None
5 class CreateLinkedList:
6     def __init__(self):
7         self.head=None
8
9     def addElement(self,data):
10        new_data= Node(data)
11
12        if(self.head==None):
13            self.head=new_data
14            return
15        temp=self.head
16        while(temp.next!=None):
17            temp=temp.next
18        temp.next=new_data
19        temp=None
20        return
21
22    def removeDuplicate(self):
23        temp = self.head
24        if temp is None:
25            return
26        while temp.next is not None:
27            if temp.data == temp.next.data:
28                new = temp.next.next
29                temp.next = None
30                temp.next = new
31            else:
32                temp = temp.next
33        temp=None
34        return
35
36
37
38    def check(self):
39        temp=self.head
```

```
40         data1=temp.next.next.next.data
41         print(data1)
42
43     def print1(self):
44         temp=self.head
45         while(temp!=None):
46             print(temp.data,end=" ")
47             temp=temp.next
48
49
50
51 list1=CreateLinkList()
52 list1.addElement(1)
53 list1.addElement(1)
54 list1.addElement(6)
55 list1.addElement(4)
56 list1.addElement(5)
57 print("Created Linked List",end=" ")
58 list1.print1()
59 print()
60 list1.removeDuplicate()
61 print("after the removing duplicates",end=" ")
62 list1.print1()
```

Created Linked List 1 1 6 4 5

after the removing duplicates 1 6 4 5

**Given the head of a linked list,  
remove the nth node from the end of  
the list and return its head.**

In [36]:

```
1 class Node:
2     def __init__(self,data):
3         self.data=data
4         self.next=None
5 class CreateLinkedList:
6     def __init__(self):
7         self.head=None
8
9     def addElement(self,data):
10        new_data = Node(data)
11        if(self.head==None):
12            self.head=new_data
13            return
14        temp=self.head
15        while(temp.next!=None):
16            temp=temp.next
17        temp.next=new_data
18        temp=None
19        return
20    def length(self):
21        count=0
22        temp=self.head
23        while(temp!=None):
24            count+=1
25            temp=temp.next
26        return count
27
28    def Removenth(self,rem):
29        n=list1.length()
30        n=n-rem
31        temp=self.head
32        if(n==0):
33            self.head=temp.next
34        for i in range(n-1):
35            temp=temp.next
36        prev=temp
37        temp = temp.next
38        if (temp==None):
39            return
```

```
40         prev.next=temp.next
41         temp=None
42
43     def print1(self):
44         temp=self.head
45         while(temp!=None):
46             print(temp.data,end=" ")
47             temp=temp.next
48
49
50
51 list1=CreateLinkList()
52 list1.addElement(1)
53 list1.addElement(2)
54 list1.addElement(3)
55 list1.addElement(4)
56 print("Created Linked List",end=" ")
57 list1.print1()
58 list1.Removenth(2)
59 print()
60 print("Linked List Ofter delete 2th",end=" ")
61 list1.print1()
62
63
64
```

Created Linked List 1 2 3 4

Linked List Ofter delete 2th 1 2 4

**Given the head of a linked list, rotate the list to the right by k places.**

In [8]:

```
1 class Node:
2     def __init__(self,data):
3         self.data=data
4         self.next=None
5 class CreateLinkedList:
6     def __init__(self):
7         self.head=None
8
9     def addElement(self,data):
10        new_data = Node(data)
11        if(self.head==None):
12            self.head=new_data
13            return
14        temp=self.head
15        while(temp.next!=None):
16            temp=temp.next
17        temp.next=new_data
18        temp=None
19        return
20    def Rotate(self,s):
21        for i in range(s):
22            temp=self.head
23            while(temp.next.next!=None):
24                temp=temp.next
25            #print(temp.data)
26            last=temp.next
27            temp.next=None
28            last.next=self.head
29            self.head=last
30        return
31
32    def print1(self):
33        temp=self.head
34        while(temp!=None):
35            print(temp.data,end=" ")
36            temp=temp.next
37
38
39
```



```
40 list1=CreateLinkList()
41 list1.addElement(1)
42 list1.addElement(2)
43 list1.addElement(3)
44 list1.addElement(4)
45 print("Created Linked List",end=" ")
46 list1.print1()
47 print()
48 list1.Rotate(2)
49 print("Ofter Roatation",end=" ")
50 list1.print1()
```

Created Linked List 1 2 3 4

Ofter Roatation 3 4 1 2

**Given a linked list and two keys in it, swap nodes for two given keys. Nodes should be swapped by changing links. Swapping data of nodes may be expensive in many situations when data contains many fields.**

It may be assumed that all keys in the linked list are distinct.

In [14]:

```
1  class LinkedList(object):
2      def __init__(self):
3          self.head = None
4
5      class Node(object):
6          def __init__(self, d):
7              self.data = d
8              self.next = None
9
10     def swapNodes(self, x, y):
11         if x == y:
12             return
13         prevX = None
14         currX = self.head
15         while currX != None and currX.data != x:
16             prevX = currX
17             currX = currX.next
18         prevY = None
19         currY = self.head
20         while currY != None and currY.data != y:
21             prevY = currY
22             currY = currY.next
23
24         if currX == None or currY == None:
25             return
26
27         if prevX != None:
28             prevX.next = currY
29         else:
30             self.head = currY
31
32         if prevY != None:
33             prevY.next = currX
34         else:
35             self.head = currX
36
37         temp = currX.next
38         currX.next = currY.next
39         currY.next = temp
```

```
40     def push(self, new_data):
41
42         new_Node = self.Node(new_data)
43
44         new_Node.next = self.head
45
46         self.head = new_Node
47
48     def printList(self):
49         tNode = self.head
50         while tNode != None:
51             print(tNode.data,end=" ")
52             tNode = tNode.next
53
54
55 llist = LinkedList()
56
57 llist.push(1)
58 llist.push(2)
59 llist.push(3)
60 llist.push(4)
61 llist.push(5)
62 llist.push(6)
63 llist.push(7)
64 print("Linked list before calling swapNodes ")
65 llist.printList()
66 llist.swapNodes(4, 3)
67 print("\nLinked list after calling swapNode ")
68 llist.printList()
```

Linked list before calling swapNodes

7 6 5 4 3 2 1

Linked list after calling swapNode

7 6 5 3 4 2 1