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

In [18]:

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

    def addElement(self,data):
        new_data= Node(data)

        if(self.head==None):
            self.head=new_data
            return
        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        temp.next=new_data
    def length(self):
        count=1
        temp=self.head

        while(temp.next!=None):
            count+=1
            temp=temp.next

        return count
    def middleElement(self):
        len1=list1.length()
        even=0
        odd=0
        print("Length",len1)
        if(len1%2==0):
            even=len1//2+1
            return list1.middle(even)
        else:
            odd=len1//2+1
            return list1.middle(odd)
    def middle(self,n):
        temp=self.head
        for i in range(n-1):
            temp=temp.next
        return temp.data


    def print1(self):
        temp=self.head
        while(temp):
            print(temp.data,end=" ")
            temp=temp.next
        print()
list1=CreateLinkList()
list1.addElement(1)
list1.addElement(2)
list1.addElement(3)
list1.addElement(4)
list1.addElement(5)
print("Created Linked List",end=" ")
list1.print1()
print("Middle Element",list1.middleElement())

```

```
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]:

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

    def addElement(self,data):
        new_data= Node(data)

        if(self.head==None):
            self.head=new_data
            return
        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        temp.next=new_data
        temp=None
        return

    def removeDuplicate(self):
        temp = self.head
        if temp is None:
            return
        while temp.next is not None:
            if temp.data == temp.next.data:
                new = temp.next.next
                temp.next = None
                temp.next = new
            else:
                temp = temp.next
        temp=None
        return



    def check(self):
        temp=self.head
        data1=temp.next.next.next.data
        print(data1)

    def print1(self):
        temp=self.head
        while(temp!=None):
            print(temp.data,end=" ")
            temp=temp.next



list1=CreateLinkList()
list1.addElement(1)
list1.addElement(1)
list1.addElement(6)
list1.addElement(4)
list1.addElement(5)
print("Created Linked List",end=" ")
list1.print1()
print()
list1.removeDuplicate()
print("ofter the removing duplicates",end=" ")
list1.print1()
```

```
Created Linked List 1 1 6 4 5 
ofter 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]:

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

    def addElement(self,data):
        new_data = Node(data)
        if(self.head==None):
            self.head=new_data
            return
        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        temp.next=new_data
        temp=None
        return
    def length(self):
        count=0
        temp=self.head
        while(temp!=None):
            count+=1
            temp=temp.next
        return count

    def Removenth(self,rem):
        n=list1.length()
        n=n-rem
        temp=self.head
        if(n==0):
            self.head=temp.next
        for i in range(n-1):
            temp=temp.next
        prev=temp
        temp = temp.next
        if (temp==None):
            return
        prev.next=temp.next
        temp=None

    def print1(self):
        temp=self.head
        while(temp!=None):
            print(temp.data,end=" ")
            temp=temp.next



list1=CreateLinkList()
list1.addElement(1)
list1.addElement(2)
list1.addElement(3)
list1.addElement(4)
print("Created Linked List",end=" ")
list1.print1()
list1.Removenth(2)
print()
print("Linked List Ofter delete 2th",end=" ")
list1.print1()
```

```
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]:

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

    def addElement(self,data):
        new_data = Node(data)
        if(self.head==None):
            self.head=new_data
            return
        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        temp.next=new_data
        temp=None
        return
    def Rotate(self,s):
        for i in range(s):
            temp=self.head
            while(temp.next.next!=None):
                temp=temp.next
            #print(temp.data)
            last=temp.next
            temp.next=None
            last.next=self.head
            self.head=last
        return

    def print1(self):
        temp=self.head
        while(temp!=None):
            print(temp.data,end=" ")
            temp=temp.next



```

```
list1=CreateLinkList()
list1.addElement(1)
list1.addElement(2)
list1.addElement(3)
list1.addElement(4)
print("Created Linked List",end=" ")
list1.print1()
print()
list1.Rotate(2)
print("Ofter Roatation",end=" ")
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]:

```
class LinkedList(object):
    def __init__(self):
        self.head = None

    class Node(object):
        def __init__(self, d):
            self.data = d
            self.next = None
    def swapNodes(self, x, y):
        if x == y:
            return
        prevX = None
        currX = self.head
        while currX != None and currX.data != x:
            prevX = currX
            currX = currX.next
        prevY = None
        currY = self.head
        while currY != None and currY.data != y:
            prevY = currY
            currY = currY.next

        if currX == None or currY == None:
            return

        if prevX != None:
            prevX.next = currY
        else:
            self.head = currY

        if prevY != None:
            prevY.next = currX
        else:
            self.head = currX

        temp = currX.next
        currX.next = currY.next
        currY.next = temp

    def push(self, new_data):

        new_Node = self.Node(new_data)

        new_Node.next = self.head

        self.head = new_Node

    def printList(self):
        tNode = self.head
        while tNode != None:
            print(tNode.data,end=" ")
            tNode = tNode.next


llist = LinkedList()

llist.push(1)
llist.push(2)
llist.push(3)
llist.push(4)
llist.push(5)
llist.push(6)
llist.push(7)
print("Linked list before calling swapNodes ")
llist.printList()
llist.swapNodes(4, 3)
print("\nLinked list after calling swapNode ")
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
```