In [20]:

```python
# A
class Node:
    def __init__(self,value):
        self.value=value
        self.next=None
class LinkedList:
    def __init__(self):
        self.head=None
        self.tail=None
    def insertAtFirst(self,value):
        newnode=Node(value)
        if self.head == None:
            self.head=newnode
            self.tail=newnode
        else:
            newnode.next=self.head
            self.head=newnode
    def insertAtEnd(self,value):
        newnode=Node(value)
        x=self.tail
        if self.head == None:
            self.head=newnode
            self.tail=newnode
        else:
            self.tail.next=newnode
            self.tail=newnode
    def insertAtAfter(self,position,value):
        newnode=Node(value)
        if self.tail == None:
            self.head=newnode
            self.tail=newnode
        else:
            temp=self.head
            while temp.value != position:
                temp=temp.next
            a=temp.next
            temp.next=newnode
            newnode.next=a
    def deleteAtFirst(self):
        x=self.head
        x=x.next
        self.head=x
    def deleteAtEnd(self):
        p=self.head
        q=p.next
        while q.next != None:
            p=p.next
            q=q.next
        p.next=None
    def deleteAtAfter(self,position):
        if self.tail == None:
            pass
        elif self.head.value==position:
            self.deleteAtFirst()
        else:
            p=self.head
            q=p.next
            while q.value != position:
                p=p.next
                q=q.next
            p.next=q.next
    def Print(self):
        x=self.head
        print("Linked List:")
        while x:
            print(x.value,end=" ")
            x=x.next
a=LinkedList()
a.insertAtFirst(5)
a.insertAtFirst(7)
a.insertAtEnd(2)
a.insertAtEnd(3)
a.deleteAtFirst()
a.deleteAtEnd()
a.insertAtAfter(5,8)
a.insertAtAfter(5,9)
a.deleteAtAfter(9)
a.insertAtAfter(2,4)
a.Print()
```

```
Linked List:
5 8 2 4
```

In [21]:

```python
# B
class Node:
    def __init__(self,value):
        self.value=value
        self.next=None
class ListStack:
    def __init__(self,size):
        self.size=size
        self.top=0
        self.head=None
        self.tail=None
    def isEmpty(self):
        if self.top==0:
            return True
        else:
            return False
    def Push(self,value):
        if self.top==self.size:
            print("Stack Overflow !")
        else:
            self.top+=1
            newnode=Node(value)
            if self.head == None:
                self.head=newnode
                self.tail=newnode
            else:
                newnode.next=self.head
                self.head=newnode
    def Pop (self):
        if self.isEmpty():
            print("Stack Underflow !")
        else:
            self.top-=1
            x=self.head
            x=x.next
            self.head=x
    def Check(self):
        if self.isEmpty:
            True
        else:
            False
    def Peek(self):
        print("Peek value of stack is:",self.head.value)
    def Count(self):
        return "Number of elements in stack: "+str(self.top)
    def Print(self):
        x=self.head
        print("\nTop to down.")
        while x:
            print("Stack:|_",x.value,"_|")
            x=x.next
ob=ListStack(3)
ob.Push(7)
ob.Push(6)
ob.Push(5)
ob.Pop()
ob.Push(1)
ob.Peek()
print(ob.Count())
ob.Print()
```

```
Peek value of stack is: 1
Number of elements in stack: 3

Top to down.
Stack:|_ 1 _|
Stack:|_ 6 _|
Stack:|_ 7 _|
```

In [23]:

```python
# C
class Node:
    def __init__(self,value):
        self.value=value
        self.next=None
class ListQueue:
    def __init__(self,size):
        self.size=size
        self.head=None
        self.tail=None
    def enQueue(self,value):
        if self.Count()==self.size:
            pass
        else:
            newnode=Node(value)
            x=self.tail
            if self.head == None:
                self.head=newnode
                self.tail=newnode
            else:
                self.tail.next=newnode
                self.tail=newnode
    def deQueue(self):
        if self.head==None:
            pass
        else:
            x=self.head
            x=x.next
            self.head=x
    def isEmpty(self):
        if self.Count==0:
            return True
        else:
            return False
    def Count(self):
        x=self.head
        count=0
        while x:
            count+=1
            x=x.next
        return count
    def Printt(self):
        x=self.head
        print("Queue:")
        while x:
            print(x.value,end=" ")
            x=x.next
ob=ListQueue(4)
ob.enQueue(1)
ob.enQueue(2)
ob.enQueue(3)
ob.enQueue(4)
ob.enQueue(5)
ob.enQueue(6)
ob.deQueue()
ob.Printt()
print("\nLength of queue is:",ob.Count())
```

```
Queue:
2 3 4
Length of queue is: 3
```

In [24]:
```
# HOME WORK (DOUBLE LINKED LIST)
class Node:
    def __init__(self,value):
        self.value=value
        self.next=None
        self.prev=None
class LinkedList:
    def __init__(self):
        self.head=None
        self.tail=None
    def insertAtFirst(self,value):
        newnode=Node(value)
        if self.head == None:
            self.head=newnode
            self.tail=newnode
        else:
            newnode.next=self.head
            self.head.prev=newnode
            self.head=newnode
    def insertAtEnd(self,value):
        newnode=Node(value)
        x=self.tail
        if self.head == None:
            self.head=newnode
            self.tail=newnode
        else:
            self.tail.next=newnode
            newnode.prev=self.tail
            self.tail=newnode
    def insertAtAfter(self,position,value):
        newnode=Node(value)
        temp=self.head
        while temp.value != position:
            temp=temp.next
        if self.tail == None:
            self.head=newnode
            self.tail=newnode
        #elif temp.next==None:

        elif self.head.next==None or temp.next==None:
            self.insertAtEnd(value)
        else:
            temp=self.head
            while temp.value != position:
                temp=temp.next
            a=temp.next
            temp.next=newnode
            newnode.prev=temp
            newnode.next=a
            a.prev=newnode
    def deleteAtFirst(self):
        x=self.head
        x=x.next
        self.head=x
        self.head.prev=None
    def deleteAtEnd(self):
        self.tail=self.tail.prev
        self.tail.next=None
    def deleteAtAfter(self,position):
        if self.tail == None:
            pass
        elif self.head.value==position:
            self.deleteAtFirst()
        else:
            p=self.head
            q=p.next
            while q.value != position:
                p=p.next
                q=q.next
            q.next.prev=p
            p.next=q.next
    def Print(self):
        x=self.head
        print("Linked List:")
        while x:
            print(x.value,end=" ")
            x=x.next
    def PrintRev(self):
        z=self.tail
        print("\nReverse list using previous node.")
        while z:
            print(z.value,end=" ")
            z=z.prev
a=LinkedList()
a.insertAtFirst(5)
a.insertAtFirst(7)
a.insertAtEnd(2)
a.insertAtEnd(3)
a.deleteAtFirst()
a.deleteAtEnd()
a.insertAtAfter(5,8)
a.insertAtAfter(5,9)
a.deleteAtAfter(9)
a.insertAtAfter(2,4)
a.Print()
a.PrintRev()
```

```
Linked List:
5 8 2 4
Reverse list using previous node.
4 2 8 5
```