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

def insertion(head,data,i=0):
    if i<0 or i>length(head):
        return head
    curr=head
    prev=None
    count=0
    while count<i:
        prev=curr
        curr=curr.next
        count=count+1
    if prev is None:
        newNode=Node(data)
        newNode.next=curr
        head=newNode
    else:
        newNode=Node(data)
        prev.next=newNode
        newNode.next=curr
    return head



c1=Node(10)
c2=Node(20)
c3=Node(30)
c4=Node(40)
c1.next=c2
c2.next=c3
c3.next=c4
printll(c1)
print(length(c1))
x=insertion(c1,200)
printll(x)
```

```
10--> 20--> 30--> 40--> None
4
200--> 10--> 20--> 30--> 40--> None
```

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

def insertion(head,data,i=0):
    if i<0 or i>length(head):
        return head
    curr=head
    prev=None
    count=0
    while count<i:
        prev=curr
        curr=curr.next
        count=count+1
    newNode=Node(data)
    if prev is None:
        newNode.next=curr
        head=newNode
    else:
        prev.next=newNode
        newNode.next=curr
    return head

def deletion(head,i=0):
    if i==0:
        head=head.next
    else:
        curr=head
        count=1
        while count<i:
            curr=curr.next
            count=count+1
        curr.next=curr.next.next
    return head



c1=Node(10)
c2=Node(20)
c3=Node(30)
c4=Node(40)
c1.next=c2
c2.next=c3
c3.next=c4
printll(c1)
print(length(c1))
x=insertion(c1,200,200)
printll(x)
y=deletion(x,length(x)-1)
printll(y)
```

```
10--> 20--> 30--> 40--> None
4
10--> 20--> 30--> 40--> None
10--> 20--> 30--> None
```

In [ ]:
```
#applications of Linkedlist
1.Used for implementing stacks and Queues
2.Dynamic memeory allocation
3.Sparse Matrix
#Real Life example:
1.Web browser hyper link
2.Image viewer
3.Music Player
4.Youtube videos
```

In [ ]:
```
#perform linear search in linkedlist--> 10->20->30->40   key=40    return 3 else return -1
#nth node problem --> 10->20->30->40 n=2 -->30
```

In [ ]:
```
#Stack --> A linear Datastructure that will follow the principle of LIFO(Last in first out)
--> whatever is coming last moved out first
In stack Insertion and Deletion both are done at the same end (Top)
How we can implement to stack?
1.Array
2.linkedlist
Terminology of stack:
    push()--> Insert any element into the stack
    pop()--> delete any elemt from the stack
    peek()--> return the top value
```

In [19]:
```
#Implement stack using array
class Stack:
    def __init__(self):
        self.array=[]
    def push(self,item):
        self.array.append(item)

    def pop(self):
        if len(self.array)==0:
            print("Stack is Empty!!!")
            return
        return self.array.pop()
    def peek(self):
        if len(self.array)==0:
            print("Stack is Empty!!!")
            return
        return self.array[len(self.array)-1]
c=Stack()
c.push(10)
c.push(20)
c.push(30)
print(c.pop())

print(c.pop())
print(c.pop())
```

```
30
20
10
```

In [ ]:
```
#Implementation of Stack using Linkedlist
count=0
for Push -->
1.Create  a node
2.Make Connections
3.Change head
4.Count=count+1

for pop
1.head=head.next
2.count=count-1

for peek
1.return head.data

for checking stack is empty or not
if count==0:
    print("empty")
    pop()
    peek()
```

In [ ]:
```
#Applications of Stack
1.Evaluation of Airthmetic expression --> 1+2//22*22/22
2.Reverse the element
3.BackTracking algorithms
4.Stack memeory management and funtion calling
5.Tower of Hanoi --> Recusrion
6.Balanced Parenthesis
7.Expression Evalution --> Prefix expression  -->+ab
                       -->infix exprssion     -->a+b
                       -->Postfix expression --> ab+
Reallife Example of stack:
    1.Bangles
    2.bundles of Plates
    3.Library books
    4.Shoe boxes
```

In [ ]:
```
File Truncate Functionj --> resizing the file
```

In [21]:
```
f=open("abc.txt","a")
f.truncate(20)
f.close()
```

In [ ]:
```

```

In [ ]:
```
f=open("")
```