In [42]:
```
#Reverse a linked list in groups of given size

class Node(object):
    __slots__ = 'data', 'next'
    def __init__(self, data = None, next = None):
        self.data = data
        self.next = next
    def __repr__(self):
        return repr(self.data)

class LinkedList(object):
    def __init__(self):
        self.head = None
    def __repr__(self):
        nodes = []
        curr = self.head
        while curr:
            nodes.append(repr(curr))
            curr = curr.next
        return '[' + ', '.join(nodes) + ']'
    def prepend(self, data):
        self.head = Node(data = data,
                         next = self.head)

    def reverse(self, k = 2):
        if self.head is None:
            return

        curr = self.head
        prev = None
        nexx = [] 
        while curr is not None:
            val = 0
            
            while curr is not None and val < k:
                new_stack.append(curr.data)
                curr = curr.next
                val += 1


            while new_stack:
                


                if prev is None:
                    curr = Node(new_stack.pop())
                    self.head = prev
                else:
                    prev.next = Node(new_stack.pop())
                    prev = prev.next


        prev.next = None
        return self.head
llist = LinkedList()
llist.prepend(6)
llist.prepend(5)
llist.prepend(4)
llist.prepend(3)
llist.prepend(2)
llist.prepend(1)

print("Given linked list")
print(llist)
llist.head = llist.reverse(5)

print("Reversed Linked list")
print(llist)
```

```
Given linked list
[1, 2, 3, 4, 5, 6]
Reversed Linked list
[5, 4, 3, 2, 1, 6]
```

In [52]:
```
# Delete the elements in an linked list whose sum is equal to zero
class Node():
    def __init__(self,data):
        self.data = data
        self.next = None

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

    def append(self,data):
        new_node = Node(data)
        h = self.head
        if self.head is None:
            self.head = new_node
            return
        else:
            while h.next!=None:
                h = h.next
            h.next = new_node

    def remove_zeros_from_linkedlist(self, head):
        stack = []
        curr = head
        list = []
        while (curr):
            if curr.data >= 0:
                stack.append(curr)
            else:
                temp = curr
                sum = temp.data
                flag = False
                while (len(stack) != 0):
                    temp2 = stack.pop()
                    sum += temp2.data
                    if (sum == 0):
                        flag = True
                        list = []
                        break
                    elif sum > 0:
                        list.append(temp2)
                if not flag:
                    for i in range(len(list)):
                        stack.append(list.pop())
                stack.append(temp)
            curr = curr.next
        return [i.data for i in stack]

if __name__ == "__main__":
    l = LinkedList()

    l.append(200)
    l.append(100)
    l.append(-22)
    l.append(44)

    print(l.remove_zeros_from_linkedlist(l.head))
```

```
[200, 100, -22, 44]
```

In [55]:
```
# Merge a linked list into another linked list at alternate positions
class Node(object):
    def __init__(self, data:int):
        self.data = data
        self.next = None


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

    def push(self, new_data:int):
        new_node = Node(new_data)
        new_node.next = self.head

        self.head = new_node


    def printList(self):
        temp = self.head
        while (temp != None):
            print(temp.data)
            temp = temp.next


    def merge(self, p, q):
        p_curr = p.head
        q_curr = q.head


        while p_curr != None and q_curr != None:


            p_next = p_curr.next
            q_next = q_curr.next


            q_curr.next = p_next
            p_curr.next = q_curr


            p_curr = p_next
            q_curr = q_next
            q_head = q_curr



llist1 = LinkedList()
llist2 = LinkedList()

llist1.push(5)
llist1.push(4)
llist1.push(3)
llist1.push(2)
llist1.push(1)


for i in range(6, 3, -1):
    llist2.push(i)

print("First Linked List:")
llist1.printList()

print("Second Linked List:")
llist2.printList()


llist1.merge(p=llist1, q=llist2)

print("Modified first linked list:")
llist1.printList()

print("Modified second linked list:")
llist2.printList()
```

```
First Linked List:
1
2
3
4
5
Second Linked List:
4
5
6
7
Modified first linked list:
1
4
2
5
3
6
4
7
5
8
Modified second linked list:
```

In [58]:
```
# In an array, Count Pairs with given sum
def getPairsCount(arr, n, sum):
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            if arr[i] + arr[j] == sum:
                count += 1
    return count
arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6
print("Count of pairs is",
      getPairsCount(arr, n, sum))
```

```
Count of pairs is 3
```

In [20]:
```
# Find duplicates in an array
list = [1,2,3,4,5,3,4,7,8,9,3,4,6,7,1,3,5]
new = []
for a in list:
    if a not in new:
        new.append(a)
print(new)
```

```
[1, 3, 4, 5, 7]
```

In [23]:
```
# Find the Kth largest and Kth smallest number in an array
def kthSmallest(arr, n, k):
    arr.sort()
    return arr[k-1];
if __name__=="__main__":
    arr = [44,85,94,88,54]
    n = len(arr)
    k = 2
    print('The smallest element is',
          kthSmallest(arr, n, k))
```

```
The smallest element is 54
```

In [25]:
```
# Move all the negative elements to one side of the array
def rearrange(arr, n ) :
    j = 0
    for i in range(0, n) :
        if (arr[i] < 0) :
            temp = arr[i]
            arr[i] = arr[j]
            arr[j]= temp
            j = j + 1
    print(arr)
arr = [-1,2,-4,6,-3,9,-4,6,-8,10]
n = len(arr)
rearrange(arr, n)
```

```
[-1, -4, -3, -4, -8, 0, 6, 2, 10]
```

In [29]:
```
# Reverse a string using a stack data structure
def create():
    stack=[]
    return stack
def size(stack):
    return len(stack)
def isEmpty(stack):
    if size(stack) == 0:
        return True
def push(stack,item):
    stack.append(item)
def pop(stack):
    if isEmpty(stack): return
    return stack.pop()
def reverse(string):
    n = len(string)
    stack = create()
    for i in range(0,n,1):
        push(stack,string[i])
    string=""
    for i in range(0,n,1):
        string+=pop(stack)
    return string
string="Data"
string = reverse(string)
print('Reversed word is' + string)
```

```
Reversed Word is)aru5
```

In [28]:
```
# Implement a queue using the stack data structure
class Queue:
    def __init__(self):
        self.s1 = []
        self.s2 = []


    def enQueue(self, x):
        while len(self.s1) != 0:
            self.s2.append(self.s1[-1])
            self.s1.pop()
        self.s1.append(x)
        while len(self.s2) != 0:
            self.s1.append(self.s2[-1])
            self.s2.pop()
    def deQueue(self):
        if len(self.s1) == 0:
            print("Q is Empty")
        x = self.s1[-1]
        self.s1.pop()
        return x
if __name__ == '__main__':
    q = Queue()
    q.enQueue(1)
    q.enQueue(2)
    q.enQueue(3)

    print(q.deQueue())
    print(q.deQueue())
    print(q.deQueue())
```

```
1
2
3
```

In [29]:
```
# Evaluate a postfix expression using stack
class Evaluate:
    def __init__(self, capacity):
        self.top = -1
        self.capacity = capacity
        self.array = []
    def isEmpty(self):
        return True if self.top == -1 else False
    def peek(self):
        return self.array[-1]
    def pop(self):
        if not self.isEmpty():
            self.top -= 1
            return self.array.pop()
        else:
            return "$"
    def push(self, op):
        self.top += 1
        self.array.append(op)
    def evaluatePostfix(self, exp):
        for i in exp:
            if i.isdigit():
                self.push(i)
            else:
                val1 = self.pop()
                val2 = self.pop()
                self.push(str(eval(val2 + i + val1)))

        return int(self.pop())
exp = "231*+9-"
obj = Evaluate(len(exp))
print ("postfix evaluation: %s"%(obj.evaluatePostfix(exp)))
```

```
postfix evaluation: -4
```

In [ ]:
```

```