Assignment-2: Linear Data Structures

1 -: Delete the elements in an linked list whose sum is equal to zero.

In [ ]:
```
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:
                 if len(list) > 0:
                     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(4)
 l.append(6)
 l.append(-10)
 l.append(8)
 l.append(9)
 l.append(10)
 l.append(-19)
```

```
 l.append(10)
 l.append(-18)
 l.append(20)
 l.append(25)

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

```
2
```

In [ ]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None

    def reverse(self, head, k):

        if head == None:
            return None
        current = head
        next = None
        prev = None
        count = 0
        while(current is not None and count < k):
            next = current.next
            current.next = prev
            prev = current
            current = next
            count += 1
        if next is not None:
            head.next = self.reverse(next, k)

        return prev

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

    def printList(self):
        temp = self.head
        while(temp):
            print(temp.data,end=' ')
            temp = temp.next


llist = LinkedList()
llist.push(9)
llist.push(8)
llist.push(7)
llist.push(6)
llist.push(5)
llist.push(4)
llist.push(3)
llist.push(2)
llist.push(1)

print("Given linked list")
llist.printList()
llist.head = llist.reverse(llist.head, 3)

print ("\nReversed Linked list")
```

```
llist.printList()
```

3-: Merge a linked list into another linked list at alternate positions.

In [ ]:
```
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(3)
llist1.push(2)
llist1.push(1)
llist1.push(0)
for i in range(8, 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()
```

4-: In an array, Count Pairs with given sum.

In [ ]:
```
l = int(input("Enter the size of array -: "))
arr = []
for i in range(l):
    ele = int(input("Enter the element of array : "))
    arr.append(ele)
print(arr)
count = 0
b = int(input("Enter the Desire sum :-"))
for j in range(0,len(arr)):
    for k in range(j+1,len(arr)):
        if arr[j]+arr[k]==b:
            count = count + 1
            print("Pairs -:",arr[j],arr[k])
print("Count Paires -:" ,count)
```

5-: Find duplicates in an array.

In [ ]:
```
l = int(input("Enter the size of array -: "))
arr = []
for i in range(l):
    ele = int(input("Enter the element of array : "))
    arr.append(ele)
print(arr)
for j in range(0,len(arr)):
    for k in range(j+1,len(arr)):
        if arr[j]==arr[k]:
            print(arr[k])
```

6-: Find the Kth largest and Kth smallest number in an array.

In [ ]:
```
## Take a user input from user to print list item.
l = int(input("Enter the size of array -: "))
arr = []
for i in range(l):
    ele = int(input("Enter the element of array : "))
    arr.append(ele)
print(arr)
## Main logic started from here.
l1 = arr[0]
for j in range(0,len(arr)):
    for k in range(j+1,len(arr)):
        if l1<arr[k]:
            tem = arr[k]
            arr[k] = l1
            l1 = tem
print("Kth largest -:",l1)
print("Kth smallest -:",arr[k])
```

7-: Move all the negative elements to one side of the array.

In [ ]:
```
## Take a user input from user to print list item.
l = int(input("Enter the size of array -: "))
arr = []
for i in range(l):
    ele = int(input("Enter the element of array : "))
    arr.append(ele)
print(arr)
## Main logic started from here.
n = []
p = []
for k in range(0,len(arr)):
        if arr[k]<0:
            n.append(arr[k])
        else:
            p.append(arr[k])
print(n+p)
```

8-: Reverse a string using a stack data structure.

In [ ]:
```
s = input("Enter the String :- ")
stack = []
for i in range(0,len(s)):
    stack.append(s[i])
for i in range(0,len(stack)):print(stack.pop())
```

9-: Evaluate a postfix expression using stack.

In [ ]:
```
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: %d"%(obj.evaluatePostfix(exp)))
```

10-: Implement a queue using the stack data structure

In [ ]:
```
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())
```