**LAB Task 6**

Name: Omerullah Ansari

ID: 65584

**Task 1**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def reverse(self):

prev = None

current = self.head

while(current is not None):

next = current.next

current.next = prev

prev = current

current = next

self.head = 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(20)

llist.push(4)

llist.push(15)

llist.push(85)

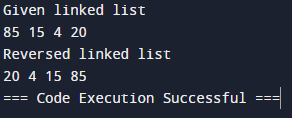
print ("Given linked list")

llist.printList()

llist.reverse()

print ("\nReversed linked list")

llist.printList()



**Task 2**

class Node:

def \_\_init\_\_(self, x):

self.data = x

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

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

def detectLoop(self):

s = set()

temp = self.head

while (temp):

if (temp in s):

return True

s.add(temp)

temp = temp.next

return False

llist = LinkedList()

llist.push(20)

llist.push(4)

llist.push(15)

llist.push(10)

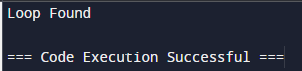
llist.head.next.next.next.next = llist.head

if(llist.detectLoop()):

print("Loop Found")

else:

print("No Loop ")



**Task 3**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def deleteNode(self, key):

temp = self.head

if (temp is not None):

if (temp.data == key):

self.head = temp.next

temp = None

return

while(temp is not None):

if temp.data == key:

break

prev = temp

temp = temp.next

if(temp == None):

return

prev.next = temp.next

temp = None

def printList(self):

temp = self.head

while(temp):

print(temp.data , end = ' ')

temp = temp.next

def removeDuplicates(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

return self.head

llist = LinkedList()

llist.push(35)

llist.push(30)

llist.push(12)

llist.push(10)

llist.push(10)

llist.push(10)

print ("Created Linked List: ")

llist.printList()

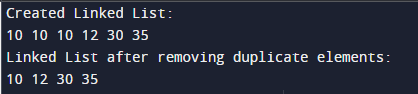
print()

print("Linked List after removing",

"duplicate elements:")

llist.removeDuplicates()

llist.printList()



**Task 4**

import heapq

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def merge\_k\_lists(lists):

heap = []

for head in lists:

if head:

heapq.heappush(heap, (head.val, head))

dummy = ListNode(0)

current = dummy

while heap:

val, node = heapq.heappop(heap)

current.next = node

current = current.next

if node.next:

heapq.heappush(heap, (node.next.val, node.next))

return dummy.next

**Task 5**

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def separate\_odd\_even(head):

if not head:

return None

even\_head = ListNode()

odd\_head = ListNode()

even\_current = even\_head

odd\_current = odd\_head

current = head

is\_even = True

while current:

if is\_even:

even\_current.next = current

even\_current = even\_current.next

else:

odd\_current.next = current

odd\_current = odd\_current.next

is\_even = not is\_even

current = current.next

even\_current.next = odd\_head.next

odd\_current.next = None

return even\_head.next

def print\_linked\_list(head):

current = head

while current:

print(current.val, end=" ")

current = current.next

print(" ")

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

head.next.next.next = ListNode(4)

head.next.next.next.next = ListNode(5)

head.next.next.next.next.next = ListNode(6)

head.next.next.next.next.next.next = ListNode(7)

print("Original list:")

print\_linked\_list(head)

rearranged\_head = separate\_odd\_even(head)

print("Rearranged list:")

print\_linked\_list(rearranged\_head)

