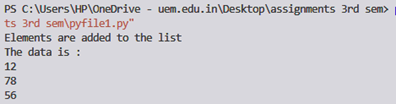
DSA ASSIGNMENT 5



class Node:

 def \_\_init\_\_(self, my\_data):

self.data = my\_data

self.next = None

class circularLinkedList:

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

self.head = None

def add\_data(self, my\_data):

ptr\_1 = Node(my\_data)

temp = self.head

ptr\_1.next = self.head

if self.head is not None:

while(temp.next != self.head):

temp = temp.next

temp.next = ptr\_1

else:

ptr\_1.next = ptr\_1

self.head = ptr\_1

def print\_it(self):

temp = self.head

if self.head is not None:

while(True):

print("%d" %(temp.data)),

temp = temp.next

if (temp == self.head):

break

my\_list = circularLinkedList()

print("Elements are added to the list ")

my\_list.add\_data (56)

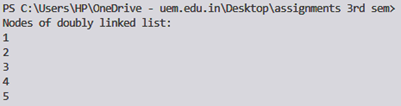
my\_list.add\_data (78)

my\_list.add\_data (12)

print("The data is : ")

my\_list.print\_it()



class Node:

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

self.data = data;

self.previous = None;

self.next = None;

class DoublyLinkedList:

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

self.head = None;

self.tail = None;

def addNode(self, data):

newNode = Node(data);

if(self.head == None):

self.head = self.tail = newNode;

self.head.previous = None;

self.tail.next = None;

else:

self.tail.next = newNode;

newNode.previous = self.tail;

self.tail = newNode;

self.tail.next = None;

def display(self):

current = self.head;

if(self.head == None):

print("List is empty");

return;

print("Nodes of doubly linked list: ");

while(current != None):

print(current.data),;

current = current.next;

dList = DoublyLinkedList();

dList.addNode(1);

dList.addNode(2);

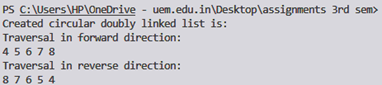
dList.addNode(3);

dList.addNode(4);

dList.addNode(5);

dList.display();



class Node:

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

self.data = data

self.next = None

self.prev = None

def insertEnd(value):

global start

if (start == None):

new\_node = Node(0)

new\_node.data = value

new\_node.next = new\_node.prev = new\_node

start = new\_node

return

last = (start).prev

new\_node = Node(0)

new\_node.data = value

new\_node.next = start

(start).prev = new\_node

new\_node.prev = last

last.next = new\_node

def insertBegin(value):

global start

last = (start).prev

new\_node = Node(0)

new\_node.data = value

new\_node.next = start

new\_node.prev = last

last.next = (start).prev = new\_node

start = new\_node

def insertAfter(value1, value2):

global start

new\_node = Node(0)

new\_node.data = value1

temp = start

while (temp.data != value2):

temp = temp.next

next = temp.next

temp.next = new\_node

new\_node.prev = temp

new\_node.next = next

next.prev = new\_node

def display():

global start

temp = start

print("Traversal in forward direction:")

while (temp.next != start):

print(temp.data, end=" ")

temp = temp.next

print(temp.data)

print("Traversal in reverse direction:")

last = start.prev

temp = last

while (temp.prev != last):

print(temp.data, end=" ")

temp = temp.prev

print(temp.data)

if \_\_name\_\_ == '\_\_main\_\_':

global start

start = None

insertEnd(5)

insertBegin(4)

insertEnd(7)

insertEnd(8)

insertAfter(6, 5)

print("Created circular doubly linked list is: ")

display()

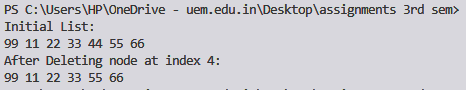


class Node:

def \_\_init\_\_(self, new\_data):

self.data = new\_data

self.next = None

 self.prev = None

def Length(head):

current = head

count = 0

if (head == None):

return 0

else:

while(True):

current = current.next

count = count + 1

if (current == head):

break;

return count

def Insert(head, data):

current = head

newNode = Node(0)

if (newNode == None):

print("\nMemory Error\n")

return None

newNode.data = data

if (head == None):

newNode.next = newNode

head = newNode

return head

else:

while (current.next != head):

current = current.next

newNode.next = head

current.next = newNode

return head

def Display(head):

current = head

if (head == None):

print("\nDisplay List is empty\n")

return

else:

while (True):

print(current.data, end=" ")

current = current.next

if (current == head):

break

def DeleteAtPosition(head, index):

len = Length(head)

count = 1

previous = head

next = head

if (head == None):

print("\nDelete Last List is empty")

return None

if (index >= len or index < 0):

print("\nIndex is not Found")

return None

if (index == 0):

head = DeleteFirst(head)

return head

while (len > 0):

if (index == count):

previous.next = next.next

return head

previous = previous.next

next = previous.next

len = len - 1

count = count + 1

return head

head = None

head = Insert(head, 99)

head = Insert(head, 11)

head = Insert(head, 22)

head = Insert(head, 33)

head = Insert(head, 44)

head = Insert(head, 55)

head = Insert(head, 66)

print("Initial List: ")

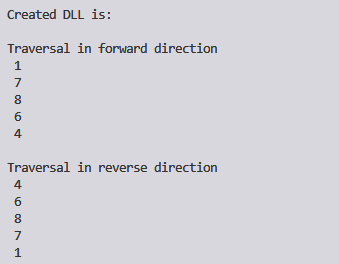
Display(head)

print("\nAfter Deleting node at index 4: ")

head = DeleteAtPosition(head, 4)

Display(head)





class Node:

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

self.data = data

self.next = None

self.prev = None

class DoublyLinkedList:

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

self.head = None

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

if self.head is not None:

self.head.prev = new\_node

self.head = new\_node

def insertAfter(self, prev\_node, new\_data):

if prev\_node is None:

print("the given previous node cannot be NULL")

return

new\_node = Node(new\_data)

new\_node.next = prev\_node.next

prev\_node.next = new\_node

new\_node.prev = prev\_node

if new\_node.next:

new\_node.next.prev = new\_node

def append(self, new\_data):

new\_node = Node(new\_data)

if self.head is None:

self.head = new\_node

return

last = self.head

while last.next:

last = last.next

last.next = new\_node

new\_node.prev = last

return

def printList(self, node):

print("\nTraversal in forward direction")

while node:

print(" {}".format(node.data))

last = node

node = node.next

print("\nTraversal in reverse direction")

while last:

print(" {}".format(last.data))

last = last.prev

if \_\_name\_\_ == "\_\_main\_\_":

llist = DoublyLinkedList()

llist.append(6)

llist.push(7)

llist.push(1)

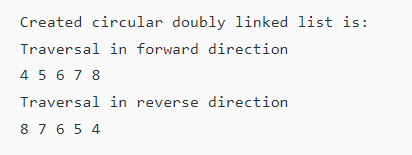
llist.append(4)

llist.insertAfter(llist.head.next, 8)

print("Created DLL is: ")

llist.printList(llist.head)



class Node:

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

self.data = data

self.next = None

self.prev = None

def insertAfter(value1, value2):

global start

new\_node = Node(0)

new\_node.data = value1

temp = start

while (temp.data != value2):

temp = temp.next

next = temp.next

temp.next = new\_node

new\_node.prev = temp

new\_node.next = next

next.prev = new\_node

def display():

global start

temp = start

print("Traversal in forward direction:")

while (temp.next != start):

print(temp.data, end=" ")

temp = temp.next

print(temp.data)

print("Traversal in reverse direction:")

last = start.prev

temp = last

while (temp.prev != last):

print(temp.data, end=" ")

temp = temp.prev

print(temp.data)

if \_\_name\_\_ == '\_\_main\_\_':

global start

start = None

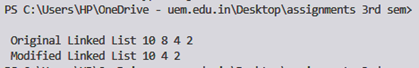
insertAfter(6, 5)

print("Created circular doubly linked list is: ")

display()



import gc

class Node:

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

self.data = data

self.next = None

self.prev = None

class DoublyLinkedList:

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

self.head = None

def deleteNode(self, dele):

if self.head is None or dele is None:

return

if self.head == dele:

self.head = dele.next

if dele.next is not None:

dele.next.prev = dele.prev

if dele.prev is not None:

dele.prev.next = dele.next

gc.collect()

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

if self.head is not None:

self.head.prev = new\_node

self.head = new\_node

def printList(self, node):

while(node is not None):

print(node.data,end=' ')

node = node.next

dll = DoublyLinkedList()

dll.push(2);

dll.push(4);

dll.push(8);

dll.push(10);

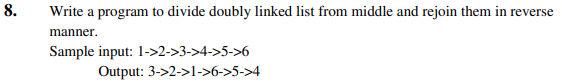
print ("\n Original Linked List",end=' ')

dll.printList(dll.head)

dll.deleteNode(dll.head.next)

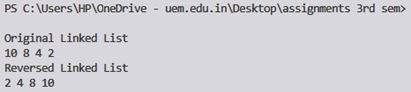
print("\n Modified Linked List",end=' ')

dll.printList(dll.head)





class Node:

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

self.data = data

self.next = None

self.prev = None

class DoublyLinkedList:

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

self.head = None

def reverse(self):

temp = None

current = self.head

while current is not None:

temp = current.prev

current.prev = current.next

current.next = temp

current = current.prev

if temp is not None:

self.head = temp.prev

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

if self.head is not None:

self.head.prev = new\_node

self.head = new\_node

def printList(self, node):

while(node is not None):

print(node.data, end=' ')

node = node.next

if \_\_name\_\_ == "\_\_main\_\_":

dll = DoublyLinkedList()

dll.push(2)

dll.push(4)

dll.push(8)

dll.push(10)

print("\nOriginal Linked List")

dll.printList(dll.head)

dll.reverse()

print("\nReversed Linked List")

dll.printList(dll.head)