|  |  |
| --- | --- |
|  | **Manav Rachna University** |
| **Lab Assignment 06** |
| **Subject:** Analysis and Design of Algorithms **Subject Code**: **CSH326**  **Semester: IV** | |

**Learning Objective:** The objective is to gain a practical understanding of recursion's applicability to linked lists.

**Learning Outcome:** Students will explore and implement recursive algorithms for common linked list operations, including traversing, searching, modifying, and creating linked lists.

All programs should be using recursion

1.Program to Print Linked List Forward and Reverse using recursion

Code:

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)

        if not self.head:

            self.head = new\_node

            return

        temp = self.head

        while temp.next:

            temp = temp.next

        temp.next = new\_node

    def print\_forward(self, node):

        if not node:

            return

        print(node.data, end=" ")

        self.print\_forward(node.next)

    def print\_reverse(self, node):

        if not node:

            return

        self.print\_reverse(node.next)

        print(node.data, end=" ")

ll = LinkedList()

ll.append(1)

ll.append(2)

ll.append(3)

ll.append(4)

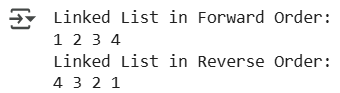
print("Linked List in Forward Order:")

ll.print\_forward(ll.head)

print("\nLinked List in Reverse Order:")

ll.print\_reverse(ll.head)

Output:



2.Program to Insert a Node at the Beginning and End .

Code:

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

def insert\_at\_beginning(head, data):

    if head is None:

        return Node(data)

    new\_node = Node(data)

    new\_node.next = head

    return new\_node

def insert\_at\_end(head, data):

    if head is None:

        return Node(data)

    head.next = insert\_at\_end(head.next, data)

    return head

def print\_list(head):

    if head is None:

        print("None")

        return

    print(head.data, "->", end=" ")

    print\_list(head.next)

head = None

head = insert\_at\_beginning(head, 3)

head = insert\_at\_beginning(head, 2)

head = insert\_at\_beginning(head, 1)

print("After inserting at the beginning:")

print\_list(head)

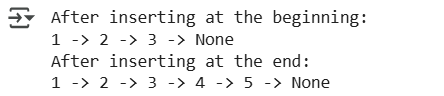
head = insert\_at\_end(head, 4)

head = insert\_at\_end(head, 5)

print("After inserting at the end:")

print\_list(head)

Output:



3.Program to merge two sorted list .

Code:

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

def merge\_sorted\_lists(l1, l2):

    if l1 is None:

        return l2

    if l2 is None:

        return l1

    if l1.data < l2.data:

        l1.next = merge\_sorted\_lists(l1.next, l2)

        return l1

    else:

        l2.next = merge\_sorted\_lists(l1, l2.next)

        return l2

def print\_list(head):

    if head is None:

        print("None")

        return

    print(head.data, "->", end=" ")

    print\_list(head.next)

def create\_list(arr):

    if not arr:

        return None

    head = Node(arr[0])

    head.next = create\_list(arr[1:])

    return head

list1 = create\_list([1, 3, 5])

list2 = create\_list([2, 4, 6])

print("List 1:")

print\_list(list1)

print("List 2:")

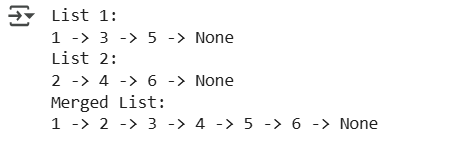
print\_list(list2)

merged\_head = merge\_sorted\_lists(list1, list2)

print("Merged List:")

print\_list(merged\_head)

Output:



4.Program to find nth node from the end in a linked list.

Code:

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

def find\_nth\_from\_end(head, n, count=[0]):

    if head is None:

        return None

    result = find\_nth\_from\_end(head.next, n, count)

    count[0] += 1

    if count[0] == n:

        return head

    return result

def print\_list(head):

    if head is None:

        print("None")

        return

    print(head.data, "->", end=" ")

    print\_list(head.next)

def create\_list(arr):

    if not arr:

        return None

    head = Node(arr[0])

    head.next = create\_list(arr[1:])

    return head

head = create\_list([1, 2, 3, 4, 5])

print("Original List:")

print\_list(head)

n = 2

nth\_node = find\_nth\_from\_end(head, n)

if nth\_node:

    print(f"The {n}th node from the end is:", nth\_node.data)

else:

    print(f"The {n}th node from the end does not exist.")

Output:

