**MUHAMMAD TAHA**

**023-23-0377**

**Task 1**

**Write a function to get the nth node from the end of the linked list.**

**Function name: int nthFromLast(int n);**

Case-1: (List Empty) Head=Null then return LIST\_EMPTY

Case-2: (List Non-Empty) Head != Null then return nth element from the end of list

Example

Input: 10 -> 20 -> 30 -> 40 -> 50, n = 2

Output: 40 (From the last, second node conatins the data 40) Code

public class Task1 {

static final int *LIST\_EMPTY* = -1;

static final int *INVALID\_INPUT* = -2;

static class Node { int data;

Node next;

Node(int data) { this.data = data;

this.next = null;

}

}

Node head;

public int nthFromLast(int n) {

if (head == null) {

return *LIST\_EMPTY*;

} if (n <= 0) {

return *INVALID\_INPUT*;

}

Node first = head;

Node second = head;

for (int i = 0; i < n; i++) { if (first == null) {

return *INVALID\_INPUT*;

}

first = first.next;

}

while (first != null) { first = first.next;

second = second.next;

}

return second.data;

}

public void add(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

} else {

Node temp = head; while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

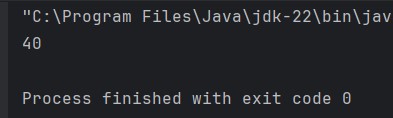
}

}

public static void main(String[] args) { Task1 list = new Task1(); list.add(10); list.add(20); list.add(30); list.add(40); list.add(50);

System.*out*.println(list.nthFromLast(2)); // Output: 40 }

}



Task 2

Write a function to sort the given single linked list. (Don’t swap the data present in the nodes, swap the nodes itself.)

Function name: void sort();

Case-1: (List Empty) Head=Null then return LIST\_EMPTY

Case-2: (List Non-Empty) Head != Null then swap the nodes to sort them

Example

Input: 50 -> 40 -> 30 -> 20 -> 10

Output: 10 -> 20 -> 30 -> 40 -> 50 Code

public class Task2 {

static final int *LIST\_EMPTY* = -1;

static class Node { int data;

Node next;

Node(int data) { this.data = data;

this.next = null;

}

}

Node head;

public int sort() { if (head == null) {

return *LIST\_EMPTY*;

}

boolean swapped; Node ptr1;

Node lptr = null;

do { swapped = false;

ptr1 = head;

while (ptr1.next != lptr) { if (ptr1.data > ptr1.next.data) {

// Swap nodes swapNodes(ptr1, ptr1.next); swapped = true;

}

ptr1 = ptr1.next;

}

lptr = ptr1;

} while (swapped);

return 0;

}

private void swapNodes(Node node1, Node node2) {

int temp = node1.data; node1.data = node2.data;

node2.data = temp;

}

public void add(int data) { Node newNode = new Node(data); if (head == null) {

head = newNode;

} else {

Node temp = head; while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public void printList() { Node temp = head; while (temp != null) {

System.*out*.print(temp.data + " -> "); temp = temp.next;

}

System.*out*.println("null");

}

public static void main(String[] args) { Task2 list = new Task2(); list.add(50); list.add(40); list.add(30); list.add(20); list.add(10);

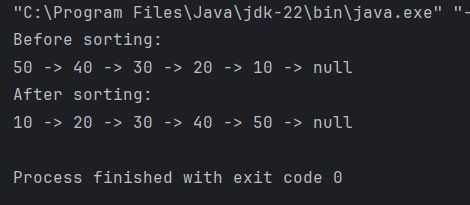
System.*out*.println("Before sorting:"); list.printList();

list.sort();

System.*out*.println("After sorting:"); list.printList();

}

}



Task 3

Write a function to reverse the single linked list.

Function name: void reverse();

Case-1: (List Empty) Head=Null then return LIST\_EMPTY

Case-2: (List Non-Empty) Head != Null then reverse the list

Example

Input: 50 -> 40 -> 30 -> 20 -> 10

Output: 10 -> 20 -> 30 -> 40 -> 50 Code

public class Task3 {

static final int *LIST\_EMPTY* = -1;

static class Node { int data;

Node next;

Node(int data) { this.data = data;

this.next = null;

}

}

Node head;

public int reverse() { if (head == null) {

return *LIST\_EMPTY*;

}

Node previous = null;

Node current = head;

Node next = null;

while (current != null) { next = current.next; current.next = previous; previous = current;

current = next;

}

head = previous;

return 0;

}

public void add(int data) {

Node newNode = new Node(data);

if (head == null) { head = newNode;

} else {

Node temp = head; while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public void printList() { Node temp = head; while (temp != null) { System.*out*.print(temp.data + " -> "); temp = temp.next;

}

System.*out*.println("null");

}

public static void main(String[] args) { Task3 list = new Task3(); list.add(50); list.add(40); list.add(30); list.add(20); list.add(10);

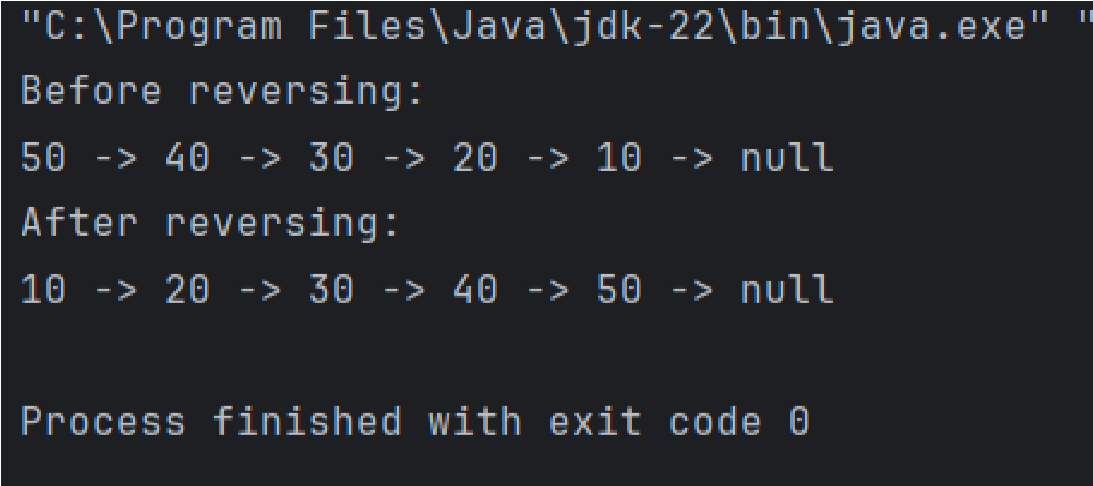
System.*out*.println("Before reversing:"); list.printList();

list.reverse();

System.*out*.println("After reversing:"); list.printList();

}

}



Task 4

Write a function to remove the duplicates data present in the single linked list.

Function name: void removeDuplicates();

Case-1: (List Empty) Head=Null then return LIST\_EMPTY

Case-2: (List Non-Empty) Head != Null then remove duplicate elements

Example

Input: 5 -> 3 -> 4 -> 5 -> 2 -> 1 -> 4 -> 5 -> 3

Output: 5 -> 3 -> 4 -> 2 -> 1 Code

import java.util.HashSet;

public class Task4 {

static final int *LIST\_EMPTY* = -1;

static class Node { int data;

Node next;

Node(int data) { this.data = data;

this.next = null;

}

}

Node head;

public int removeDuplicates() {

if (head == null) {

return *LIST\_EMPTY*;

}

HashSet<Integer> seen = new HashSet<>();

Node current = head;

Node previous = null;

while (current != null) {

if (seen.contains(current.data)) {

previous.next = current.next;

} else {

seen.add(current.data);

previous = current;

}

current = current.next;

}

return 0;

}

public void add(int data) {

Node newNode = new Node(data);

if (head == null) { head = newNode;

} else {

Node temp = head; while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public void printList() { Node temp = head; while (temp != null) {

System.*out*.print(temp.data + " -> "); temp = temp.next;

}

System.*out*.println("null");

}

public static void main(String[] args) { Task4 list = new Task4();

list.add(5); list.add(3); list.add(4); list.add(5); list.add(2); list.add(1); list.add(4); list.add(5); list.add(3);

System.*out*.println("Before removing duplicates:"); list.printList();

list.removeDuplicates();

System.*out*.println("After removing duplicates:"); list.printList();

}

}

