DataStructures and algorithms

Lab 3

Name:Pranjal bajaj

Cms:023-21-0022

Task 1

addToFront: Adds a new node with the given data to the front of the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

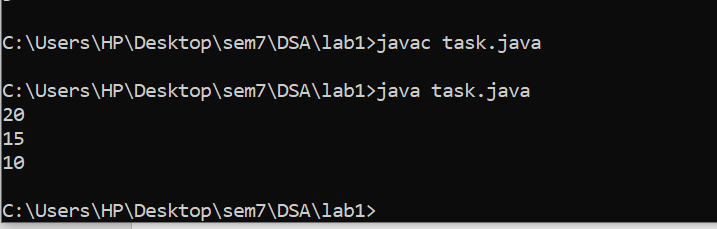
            ll.creation(15);

            ll.creation(20);

            ll.traverser();

    }

}



getFrontItem: Returns the data stored in the first node (front) of the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            // while(temp.next != null){

                System.out.println(temp.data + " ");

                // temp = temp.next;

            // }

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

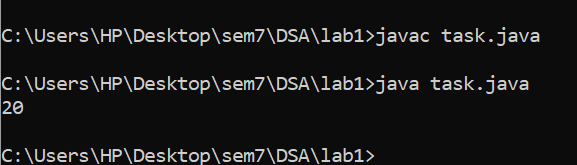
            ll.creation(15);

            ll.creation(20);

            ll.traverser();

    }

}



removeFrontItem: Removes the first node (front) from the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void removefrontitem(){

            head = head.next;

            Node temp = head;

            System.out.println("After removal of first item");

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

        }}

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

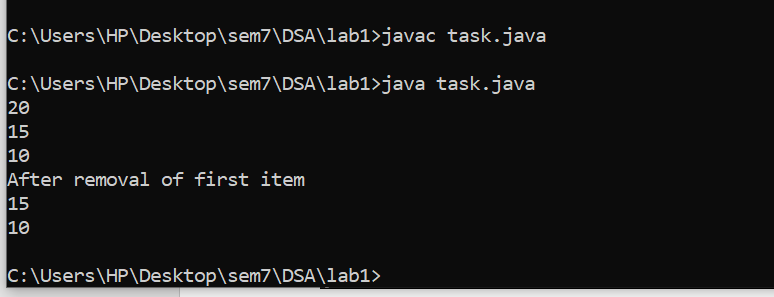
            ll.creation(20);

            ll.traverser();

            ll.removefrontitem();

    }

}



addToBack: Adds a new node with the given data to the end (back) of the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void addtoback(){

            Node temp = head;

            while(temp.next != null){

                temp = temp.next;

            }

            System.out.println("New item added at back");

            Node newnode = new Node(5);

            temp.next = newnode;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

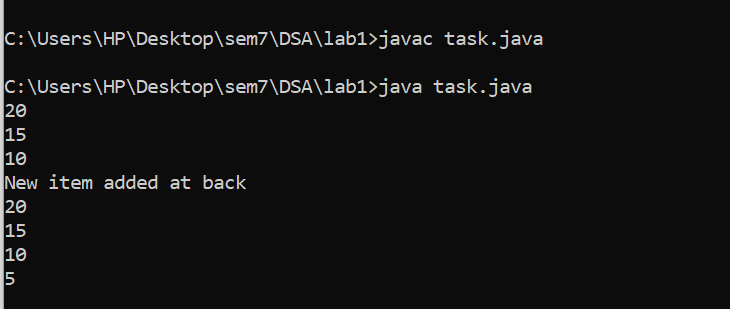
            ll.traverser();

            ll.addtoback();

            ll.traverser();

    }

}



getBackItem: Returns the data stored in the last node (back) of the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.println(temp.data + " ");

                temp = temp.next;

            }

        }

        void getbackitem(){

            Node temp = head;

            while(temp.next != null){

                temp = temp.next;

            }

            System.out.println(" item  at back is "+ temp.data);

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

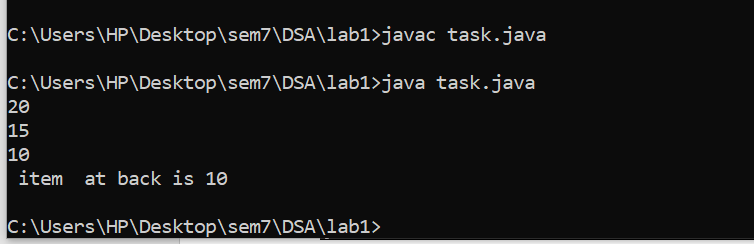
            ll.traverser();

            ll.getbackitem();

            // ll.traverser();

    }

}



removeBackItem: Removes the last node (back) from the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }

            System.out.println("");

        }

        void removebackitem(){

            Node temp = head;

            while(temp.next.next != null){

                temp = temp.next;

            }

            temp.next = null;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

            ll.creation(35);

            ll.traverser();

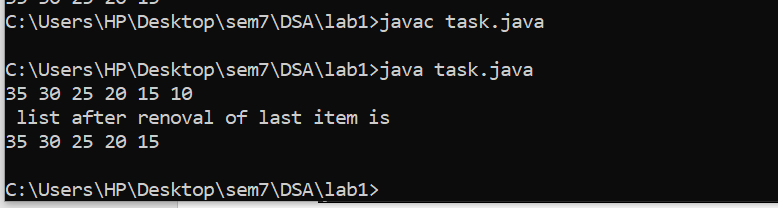
            ll.removebackitem();

            System.out.println(" list after renoval of last item is ");

            ll.traverser();

    }

}



find: Checks if the given key is present in the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }

            System.out.println("");

        }

        void find(int data){

            Node temp = head;

            while(temp.next != null){

                if(temp.data == data){

                    System.out.println(temp.data +" founded ");

                }

             temp = temp.next;

            }

            // head = head.next;

            // Node temp = head;

            // System.out.println("After removal of first item");

            // while(temp != null){

            //     System.out.println(temp.data + " ");

            //     temp = temp.next;

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

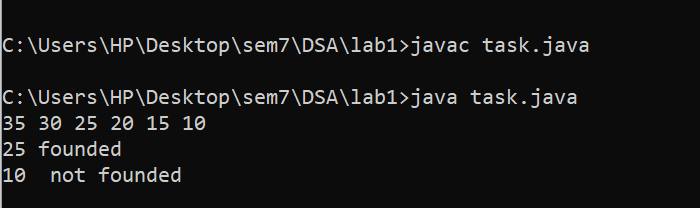
            ll.creation(35);

            ll.traverser();

            ll.find(13);

    }

}



Remove: Removes the node with the given key from the linked list, if present.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }

            System.out.println("");

        }

        void find(int data){

            Node temp = head ;

            while (temp != null && temp.next != null) {

            if (temp.next.data == data) {

                temp.next = temp.next.next;

                return;

            }

            temp = temp.next;

            }

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

            ll.creation(35);

            ll.traverser();

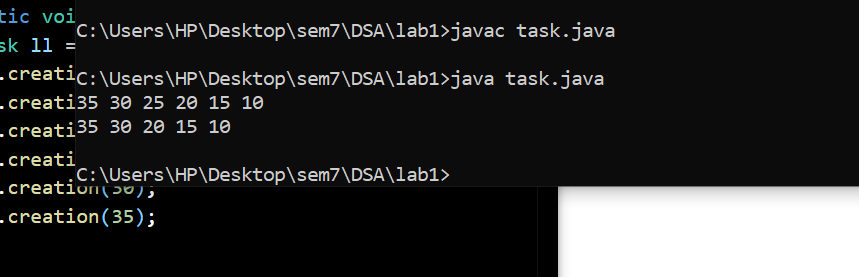
            ll.find(25);

            //System.out.println(" list after renoval of last item is ");

            ll.traverser();

    }

}



isListEmpty: Checks if the linked list is empty.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }}

            System.out.println("");

        }

    public static void main(String[] args){

            Task ll = new Task();

            // ll.creation(10);

            // ll.creation(15);

            // ll.creation(20);

            // ll.creation(25);

            // ll.creation(30);

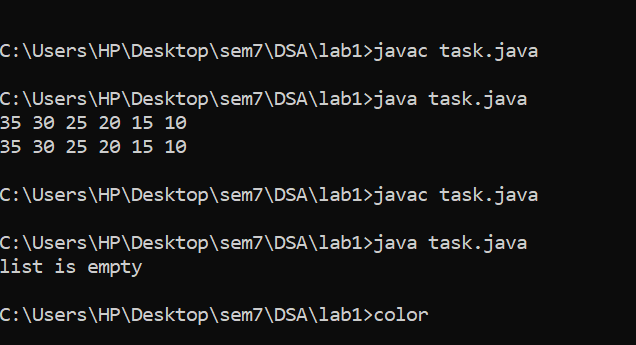
            // ll.creation(35);

            ll.traverser();

            //System.out.println(" list after renoval of last item is ");

    }

}



addKeyBeforeNode: Adds a new node with the given key before the node containing the specified data in the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }}

            System.out.println("");

        }

                    void keybeforenode(int data , int add){

                        Node temp = head, prev = head;

                        Node newNode = new Node(add);

                        if (head.data == data) {

                                newNode.next = head;

                                head = newNode;

                                return;

                            }

                            Node current = head;

                            while (current.next != null && current.next.data != data) {

                                current = current.next;

                            }

                            if (current.next == null) {

                                System.out.println("Target key not found.");

                                return;

                            }

                            newNode.next = current.next;

                            current.next = newNode;

                        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

            ll.creation(35);

            ll.traverser();

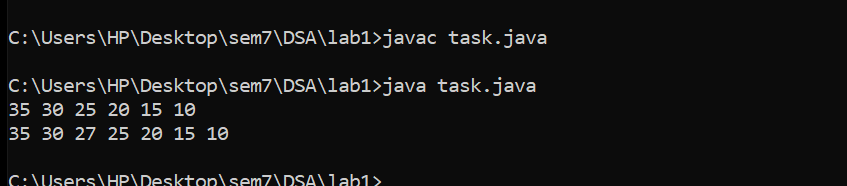
            ll.keybeforenode(25, 27);

            //System.out.println(" list after renoval of last item is ");

            ll.traverser();

    }

}



addKeyAfterNode: Adds a new node with the given key after the node containing the specified data in the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }}

            System.out.println("");

        }

                    void keyafternode(int data , int add){

                        Node temp = head, prev = head;

                        Node newNode = new Node(add);

                            Node current = head;

                            while (current!= null && current.data != data) {

                                current = current.next;

                            }

                            if (current == null) {

                                System.out.println("Target key not found.");

                                return;

                            }

                            newNode.next = current.next;

                            current.next = newNode;

                        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

            ll.creation(35);

            ll.traverser();

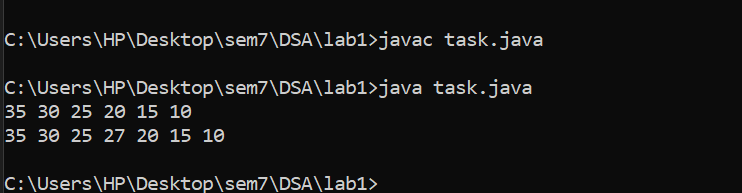
            ll.keyafternode(25, 27);

            //System.out.println(" list after renoval of last item is ");

            ll.traverser();

    }

}



printAll: Prints all the values in the linked list.

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            { newNode.next = head;

            head = newNode; }

        }

        void traverser(){

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }}

            System.out.println("");

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

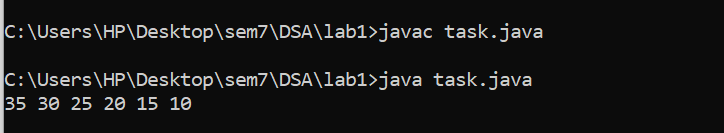
            ll.creation(30);

            ll.creation(35);

            ll.traverser();

    }

}



Task 2

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            {

                newNode.next = head;

                head = newNode;

                 }

            if(newNode.next == null){

                tail = newNode;

            }}

            void addtoback(int data){

            Node temp = tail;

            Node newnode = new Node(data);

            temp.next = newnode;

            }

            void removebackitem(){

            Node temp = head;

            while(temp != tail){

                temp = temp.next;

            }

            temp.next = null;

            tail = temp;

        }

        void traverser(){

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }}

            System.out.println("");

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

            ll.creation(35);

            ll.traverser();

            ll.addtoback(100);

            System.out.println("New item added at back");

            ll.traverser();

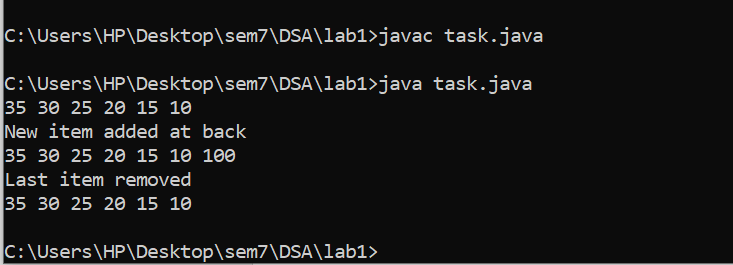
            ll.removebackitem();

            System.out.println("Last item removed");

            ll.traverser();

    }

}



Task 3

class Task{

    static class Node{

    int data;

    Node next;

    Node(int data){

        this.data = data;

        this.next = null;

    }

}

    Node head = null;

    Node tail = null;

        void creation(int data){

            Node newNode = new Node(data);

            if(head == null){

                head = newNode;

            }

            else

            {

                newNode.next = head;

                head = newNode;

                 }

            if(newNode.next == null){

                tail = newNode;

            }}

            void getsize(){

                Node temp = head;

                int size=1;

                while(temp != tail){

                    temp = temp.next;

                    size+=1;

                }

                System.out.println("total size of list is "+size);            }

            void getindex(int data){

            int count=0;

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                temp = temp.next;

                count+=1;

                if(temp.data == data){

                    System.out.println("index of "+data + " is "+count + " ");

                    return;

                }}}}

            void insertAt(int index, int data){

            Node temp = head;

            int count  = 1;

            while(count != index){

                temp = temp.next;

                count+=1;

                }

                System.out.println("New item added at "+ index);

                Node nn = new Node(data);

                nn.next = temp.next;

                temp.next = nn;

        }

        void removeatindex(int index){

            Node temp = head;

            int count  = 1;

            while(count != index){

                temp = temp.next;

                count+=1;

                }

                System.out.println(" item removed at "+ index);

                temp.next = temp.next.next;

        }

        void traverser(){

            Node temp = head;

            if(head == null){

                System.out.print("list is empty");

            }

            else{

            while(temp != null){

                System.out.print(temp.data + " ");

                temp = temp.next;

            }}

            System.out.println("");

        }

    public static void main(String[] args){

            Task ll = new Task();

            ll.creation(10);

            ll.creation(15);

            ll.creation(20);

            ll.creation(25);

            ll.creation(30);

            ll.creation(35);

            ll.traverser();

            ll.getsize();

            ll.getindex(25);

            ll.insertAt(3, 40);

             ll.traverser();

            ll.removeatindex(5);

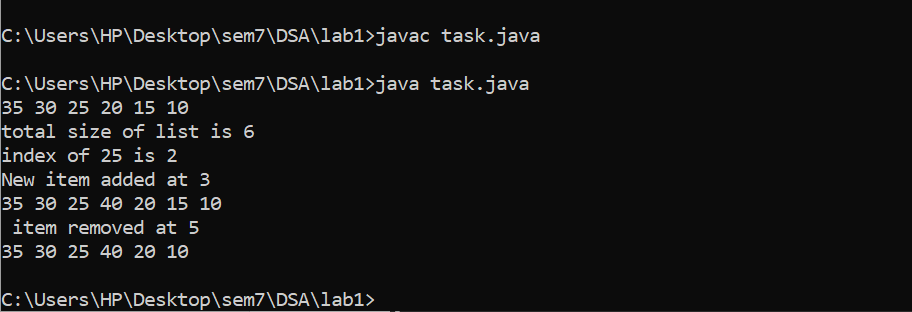
             ll.traverser();

            // ll.removebackitem();

            // System.out.println("Last item removed");

    }

}



Task 4

class task {

    class ListNode {

    int value;

    ListNode next;

    ListNode(int value) {

        this.value = value;

        this.next = null;

    }

    ListNode(int value, ListNode next) {

        this.value = value;

        this.next = next;

    }

}

    // Method to reverse the linked list

     ListNode reverse(ListNode head) {

        ListNode prev = null;

        ListNode current = head;

        ListNode next = null;

        while (current != null) {

            next = current.next;  // Store the next node

            current.next = prev;  // Reverse the link

            prev = current;       // Move prev up

            current = next;       // Move to the next node

        }

        // At the end, prev will be the new head of the reversed list

        return prev;

    }

    // Utility method to print the linked list

     void printList(ListNode head) {

        ListNode current = head;

        while (current != null) {

            System.out.print(current.value + " -> ");

            current = current.next;

        }

        System.out.println("None");

    }

    public static void main(String[] args) {

        // Create a linked list: 1 -> 2 -> 3 -> None

        ListNode head = new ListNode(1, new ListNode(2, new ListNode(3)));

        System.out.println("Original list:");

        printList(head);

        // Reverse the linked list

        head = reverse(head);

        System.out.println("Reversed list:");

        printList(head);

    }

}