IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student: N. Ravi Chandrika

Class: CSD - B Semester: Third Semester

Course Code: ACS011 Course Name: DS Laboratory

Name of the Course Faculty: Mr. A. Srikanth Faculty ID: IARE10652

Exercise Number: 7 Week Number: 7 Date: 05-11-2024

Roll Number									
0	3	9	5	1	A	6	7	R	3

S No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure	Performance in the Lab	Source Code Calculations and Graphs	Program Execution Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	7.1	Singly Linked List							
2	7.2	Linked List Cycle							
3	7.3	Remove Linked List Elements							
4	7.4	Reverse Linked List	4	2	2	4	4	4	20
5	7.5	Palindrome Linked List							
6	7.6	Middle of Linked List							
7	7.7	Convert binary number in a linked list to integer							
8									
9									
10									
11									
12									

N. Ravi Chandrika
Signature of the Student

Signature of the Faculty

7.1 Aim: a singly linked list linear data structure in which the elements are not stored in the contiguous memory and each memory is connected only to its next element using a pointer.

Code:

```
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
class LinkedLists {
    Node head;
    public void insertAtBeginning (int data) {
        Node newNode = new Node (data);
        newNode.next = head;
        head = newNode;
    }
    public void insertAtEnd (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 insertAtPosition(int data, int position) {
        Node newNode = new Node(data);
        if (position == 0) {
            insertAtBeginning(data);
        } else {
            Node temp = head;
            for (int i = 1; i < position && temp != null; i++) {
                temp = temp.next;
            }
            if (temp != null) {
                newNode.next = temp.next;
                temp.next = NewNode;
            } else {
                System.out.println("Position out of bounds");
            }
        }
    }
    public void deleteAtBeginning() {
        if (head != null) {
            head = head.next;
        }
    }
    public void deleteAtEnd() {
        if (head == null) return;
        if (head.next == null) {
```

```
        head = null;
    } else {
        Node temp = head;
        while (temp.next.next != null) {
            temp = temp.next;
        }
        temp.next = null;
    }
}
public void deleteAtPosition(int position) {
    if (head == null) return;
    if (position == 0){
        deleteAtBeginning();
    } else {
        Node temp = head;
        for (int i=1; i < position && temp.next != null; i++) {
            temp = temp.next;
        }
        if (temp.next != null) {
            temp.next = temp.next.next;
        } else {
            System.out.println("Position out of bounds");
        }
    }
}
public void traverse() {
    Node temp = head;
```

```
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    public boolean search (int key){
        Node temp = head;
        while (temp != null) {
            if (temp.data == key) {
                return true;
            }
            temp = temp.next;
        }
        return False;
    }
    public int size() {
        int count = 0;
        Node temp = head;
        while (temp != null) {
            count++;
            temp = temp.next;
        }
        return count;
    }
}
public class LinkedList {
    public static void main (String args[]) {
        LinkedLists list = new LinkedLists();
```

```
        list.insertAtBeginning(10);
        list.insertAtEnd(20);
        list.insertAtPosition(15, 1);
        System.out.println("Linked List:");
        list.traverse();
        System.out.println("Size: " + list.size());
        System.out.println("Searching for 15: " + list.search(15));
        System.out.println("Searching for 25:" + list.search(25));
        list.deleteAtPosition(1);
        System.out.println("After Deletion:");
        list.traverse();
    }
}.
```

Output:

Linked List :
10 15 20
Size : 3
Searching for 15 : True
Searching for 25 : False
After deletion :
10 20.

7.2 Aim: Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer.

Code: Class ListNode {
    int val;

```
        ListNode next;
        ListNode(int x) {
            val = x;
            next = null;
        }
}
public class LinkedListCycle {
    public boolean hasCycle(ListNode head) {
        if (head == null || head.next == null) {
            return false;
        }
        ListNode slow = head;
        ListNode fast = head.next;
        while (slow != fast) {
            if (fast == null || fast.next == null) {
                return false;
            }
            slow = slow.next;
            fast = fast.next.next;
        }
        return true;
}
public static void main(String args[]) {
    ListNode node1 = new ListNode(3);
    ListNode node2 = new ListNode(2);
    ListNode node3 = new ListNode(0);
    ListNode node4 = new ListNode(-4);
    node1.next = node2;
    node2.next = node3;
    node3.next = node4;
    node4.next = node2;
```

```
        LinkedListCycle solution = new LinkedListCycle();
        System.out.println("Cycle Detected (Example 1):" + solution.has
                                                    Cycle(node1));

        ListNode nodeA = new ListNode(1);
        ListNode nodeB = new ListNode(2);
        nodeA.next = nodeB;
        nodeB.next = nodeA;
        System.out.println("Cycle Detected (Example 2):" + solution.has
                                                    Cycle(node2));

        ListNode nodeC = new ListNode(1);
        System.out.println("Cycle Detected (Example 3):" + solution.hasCycle
                                                    (node3));

    }
}
```

Output:

Cycle Detected (Example1): True
Cycle Detected (Example 2): True
Cycle Detected (Example 3): False.

7.3 Aim: Given the head of a linked list and an integer val, remove all the nodes of the linkedlist that has Node.Val == val & return new head.

Code:

```
class ListNode {
    int val;
    ListNode next;
    ListNode (int x) {
        val = x;
        next = null;
    }
}
public class Remove_Linked_List_Elements {
    public ListNode removeElements (ListNode head, int val) {
    ListNode dummy = new ListNode(0);
```

```
        dummy.next = head;
        ListNode current = dummy;
        while (current.next != null) {
            if (current.next.val == val) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        return dummy.next;
    }
    public static void main (String args[]) {
        Remove-Linked-List-Elements = new Remove-Linked-List();
        ListNode 1 = new ListNode(1);
        ListNode 2 = new ListNode(2);
        ListNode node 3 = new ListNode(6);
        ListNode node4 = new ListNode(3);
        ListNode node5 = new ListNode(4);
        ListNode node6 = new ListNode(5);
        ListNode node7 = new ListNode(6);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        :
        node 6.next = node 7;
        printList(node1);
        ListNode result = solution.removeElements(node1, 6);
        printlist(result);
    }
    public static void printList(ListNode head) {
        ListNode current = head;
        while (current != null) {
```

```
        System.out.print (current.val + " ");
        current = current.next;
    }
    System.out.println()
  }
}
```

Input: 1 2 6 3 4 5 6

Output: 1 2 3 4 5.

7.4 Aim: Given head of a singly linked list, reverse & return the list.

Code:

```
class ListNode {
    int val;
    ListNode next;
    ListNode (int x) {
        val = x;
        next = null;
    }
}
public class ReverseLinkedList {
    public ListNode reverseList (ListNode head) {
        ListNode prev = Null;
        ListNode current = head;
        while (current != null) {
            ListNode nextTemp = current.next;
            current.next = prev;
            prev = current;
            current = nextTemp;
        }
        return prev;
    }
```

```
    public static void main(String args[]) {
        ReverseLinkedList solution = new ReverseLinkedList();
        ListNode node1 = new ListNode(1);
        ListNode node5 = new ListNode(5);
        node1.next = node2;
        node4.next = node5;
        printList(node1);
        ListNode result = solution.reverseList(node1);
        printList(result);
    }
    public static void printList(ListNode head) {
        ListNode current = head;
        while (current != null) {
            System.out.print(current.val + " ");
        }
        System.out.println();
    }
}
```

Input: 1 2 3 4 5

Output: 5 4 3 2 1.

7.5 Aim: Given the head of singly linked list, return true if it is palindrome or false otherwise.

Code:

```
class ListNode {
    int val;
    ListNode next;
    ListNode (int x) {
        val = x;
        next = null;
    }
}
```

ROLL NUMBER :

```
public class PalindromeLinkedList {
    public boolean isPalindrome (ListNode head) {
        List<Integer> values = new ArrayList<>();
        ListNode current = head;
        While (current != null) {
            values.add(current.val);
            current = current.next;
        }
        int start = 0;
        int end = values.size() - 1;
        while (start < end) {
            if (!values.get(start).equals(values.get(end))) {
                return False;
            }
            start++;
            end--;
        }
        return true;
    }
    public static void main(String args[]) {
        PalindromeLinkedList Solution = new PalindromeLinkedList();
        ListNode node1 = new ListNode(1);      # node3 = 2
        ListNode node2 = new ListNode (2);     # node4 = 1
        node1.next = node2;
        node3.next = node4;
        printList(node1);
        System.out.println(solution.isPalindrome(node1));
    }
    public static void printList (ListNode head) {
        ListNode current = head;
        While (current != null) {
```

```
        System.out.println(current.val+" ");
        current = current.next;
    }
    System.out.println();
    }
}.
```

Input: 1 2 3 4 5

Output: 5 4 3 2 1.

7.6 Aim: Given the head of a singly linked list, return middle node of list, if there are two middle nodes return second middle one.

Code:

```
class ListNode {
    int val;
    ListNode next;
    ListNode (int x) {
        val = x;
        next = null;
    }
}
public class MiddleOfTheLinkedList {
    public ListNode middleNode (ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
```

```
public void main(String args[]) {
    MiddleOfTheLinkedList solution = new MiddleOfLinkedList();
    ListNode node1 = new ListNode(1);
    ListNode node5 = new ListNode(5);
    node1.next = node2;
    node4.next = node5;
    printList(node1);
    ListNode middleNode = solution.middleNode(node1);
    printList(middleNode);
}
public static void printList(ListNode head) {
    ListNode current = head;
    while(current != null) {
        System.out.print(current.val + " ");
        current = current.next;
    }
    System.out.println();
}
}
```

Input: 1 2 3 4 5

Output: 3 4 5

7.7 Aim: Given head, which is a reference node to a singly-linked list. The value of each node in linked list is either 0 or 1. This holds the binary representation of an number.

Code:

```
class ListNode {
  int val;
  ListNode next;
  ListNode (int x) {
      val = x;
      next = null;
  }
}
public class ConvertBinary {
    public int getDecimalValue (ListNode head) {
      int result = 0;
      While (head != null) {
        result = result * 2 + head.val;
        head = head.next;
      }
      return result;
    }
    public static void main (String args[]) {
      ConvertBinary solution = new ConvertBinary();
      ListNode node1 = new ListNode (1);
      ListNode node2 = new ListNode (0);
      ListNode node3 = new ListNode (1);
      node1.next = node2;
      node2.next = node3;
      printList (node1);
      System.Out.println( solution.getDecimalValue (node1));
```

ROLL NUMBER :

```
}
public static void printList (ListNode head) {
    ListNode current = head;
    While (current != null) {
        System.out.print (current.val + " ");
        current = current.next;
    }
    System.out.println();
}
}.
```

Output: 5.