IARE

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

# LABORATORY WORK BOOK

Name of the Student : Racherla Santhosh

Class : IT-B Semester : 03

Course Code : ACSD11 Course Name : DS Laboratory

Name of the Course Faculty : Ms. K. Laxminarayanamma Faculty ID : IARE10033

Exercise Number : 07 Week Number : 07 Date : 28/10/2024

| Roll Number | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 9 | 5 | 1 | A | 1 | 2 | C | 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 | | | | | | |
| 4 | 7.4 | Reverse Linked List | | | | | | |
| 5 | 7.5 | Palindrome Linked list | | | | | | |
| 6 | 7.6 | Middle of Linked list | | | | | | |
| 7 | 7.7 | Convert Binary Number | | | | | | |
| 8 | | in a linked list to Integer. | | | | | | |
| 9 | | | 4 | 4 | 4 | 4 | 4 | 20 |
| 10 | | | | | | | | |
| 11 | | | | | | | | |
| 12 | | | | | | | | |

Signature of the Student

Signature of the Faculty

# 7. Linked List.

## 7.1 Singly Linked List :-

AIM :- Create a linked list using the following Operations :

1) Creating a Node Class :
2) Insertion at Beginning :
3) Insertion at End :
4) Insertion at Middle :
5) Update the Node
6) Deletion at Beginning
7) Deletion at End
8) Deletion at Middle
9) Remove last node
10) Linked List Traversal
11) Get Length.

PROGRAM :-

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    Node (int data) {
        this.data = data;
        this.next = null;
    }
}
class LinkedList {
    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;
        return;
    }
    Node current = head;
    while (current.next != null) {
        current = current.next;
    }
    current.next = NewNode;
}

Public void insertAtMiddle (int data, int position) {
    if (position == 0) {
        insertAtBeginning (data);
        return;
    }
    Node newNode = new Node (data.);
    Node current = head;
    for (int i=0; i< position-1; i++.) {
        if (current == null) {
            throw new IndexOutOfBoundsException
                    ("Position out of bounds");
        }
```

```
        current = current.next;
    }
    newNode.next = current.next;
    current.next = newNode;
}

Public void updateNode(int oldData, int newData){
    Node current = head;
    while (current != null) {
        if (current.data == oldData) {
            current.data = newData;
            return;
        }
        current = current.next;
    }
    throw new RuntimeException("Node not found");
}

Public void deleteAtBeginning() {
    if (head == null) return;
    head = head.next;
}

Public void deleteAtEnd() {
```

```
if (head == null) return;
if (head.next == null) {
    head = null;
    return;
}
Node current = head;
while (current.next.next != null) {
    current = current.next;
}
current.next = null;
}

Public int getLength() {
    int count = 0;
    Node current = head;
    while (current != null) {
        count++;
        current = current.next;
    }
    return count;
}
}
```

```
Public class Main {
    Public static void main(String[] args) {
        Scanner Scanner = new Scanner(System.in);
        LinkedList list = new LinkedList();
        int choice;
        do {
            System.out.println("Menu:");
            System.out.println("1. Insert at Beginning");
            System.out.println("2. Insert at End");
            System.out.println("3. Insert at Middle");
            System.out.println("4. Update Node");
            System.out.println("5. Delete at Beginning");
            System.out.println("6. Delete at End");
            System.out.println("7. Delete at Middle");
            System.out.println("8. Traverse");
            System.out.println("9. Get Length");
            System.out.println("0. Exit");
            System.out.print("Enter your choice:");
```

```
choice = Scanner.nextInt();
Switch (choice) {
 case 1:
   System.out.print ("Enter data:");
   int data1 = Scanner.nextInt();
   list.insertAtBeginning (data1);
   break;
 case 2:
   System.out.print ("Enter data:");
   int data2 = Scanner.nextInt();
   list.insertAtEnd (data2);
   break;
 case 3:
   System.out.print ("Enter data:");
   int data3 = Scanner.nextInt();
   System.out.Print ("Enter Position:");
   int position3 = Scanner.nextInt();
   list.insertAtMiddle (data3, position3);
   break;
```

```
Case 4:
    System.out.print("Enter Old data:");
    int OldData = Scanner.nextInt();
    System.out.Print("Enter new data:");
    int newData = Scanner.nextInt();
    list.updateNode(OldData, newData);
    break;
case 5:
    list.deleteAtBeginning();
    break;
case 6:
    list.deleteAtEnd();
    break;
case 7:
    System.out.print("Enter position to delete:");
    int Position7 = Scanner.nextInt();
    list.deleteAtMiddle(Position7);
    break;
Case 8:
```

```
            System.out.println ("Linked List:");
            list.traverse();
            break;
        case 9:
            System.out.println ("Length of the list: " +
                                list.getLength());
            break;
        case 0:
            System.out.println ("Exiting.....");
            break;
        default:
            System.out.println ("Invalid choice!
                                Please Try Again");
    }
} while (choice != 0);
Scanner.close();
}
}
```

RESULT :-

Menu :

1. Insert at Beginning
2. Insert at End
3. Insert at Middle
4. Update Node
5. Delete at Beginning
6. Delete at End
7. Delete at Middle
8. Traverse
9. Get Length
0. Exit

Enter your choice : 1

Enter data : 10

menu :

Enter your choice : 2

Enter data : 20

menu :

Enter your choice : 3

Enter data : 15

Enter position : 1

menu :

Enter your choice : 4

Enter old data : 15

Enter new data : 25

menu :

Enter your choice : 5

Linked List : 25 20

menu :

Enter your choice : 6

Linked List : 25

menu :

Enter your choice : 7

Enter Position to delete : 0

Linked List : (empty)

menu :

Enter your choice : 8

Linked list : (empty)

Enter your choice : 9

Length of the list : 0

Enter your choice : 0

Exiting ....

## 7.2 Linked List Cycle :-

Aim :- Write a program on the head of a linked list, Determine if the linked list has a cycle in it returns True Otherwise False.

PROGRAM :-

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

Public class Solution {
    Public boolean hasCycle (ListNode head) {
        if (head == null) return false;
        ListNode slow = head;
        ListNode fast = head;
```

```
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow == fast) {
        return true;
    }
}
return false;
}

Public Static void main (String[] args) {
ListNode head = new ListNode (3);
ListNode Second = new ListNode (2);
ListNode third = new ListNode (0);
ListNode fowth = new ListNode (-4);
head.next = Second;
Second.next = third;
third.next = fowth;
fowth.next = head;
Solution Solution = new Solution ();
```

```
        System.out.println(Solution.hasCycle(head));
    }
}
```

OUTPUT :- True.

## 7.3 Remove Linked List Elements :-

Aim :- write a Program on the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val; and return the new head.

PROGRAM :-

```
Public class LinkedListRemoveElements {
  Public ListNode removeElements(ListNode head,
                                 int val) {
    ListNode dummy = new ListNode(-1);
    dummy.next = head;
    ListNode current = dummy;
    While (current.next != null) {
```

```
        if ( current . next . data == val ) {
            current . next = current . next . next ;
        } else {
            current = current . next ;
        }
    }
    Return dummy . next ;
}
Public Static void main ( String[] args ) {
    LinkedList RemoveElements list = new LinkedList Remove
                                        Elements ();
    ListNode head = new ListNode (1);
    head . next = new ListNode (2);
    head . next . next = new ListNode (6);
    head . next . next . next = new ListNode (3);
    head . next . next . next . next = new ListNode (4);
    head . next . next . next . next . next = new ListNode (5);
    head . next . next . next . next . next . next = new
                                        ListNode (6);
```

```
    int val = 6;
    ListNode newHead = llist.removeElements(head, val);
    ListNode temp = newHead;
    while (temp != null) {
        System.out.print(temp.data + " -> ");
        temp = temp.next;
    }
    System.out.println("null");
  }
}
class ListNode {
    int data;
    ListNode next;
    ListNode(int data) {
        this.data = data;
        this.next = null;
    }
}
```

OUTPUT :-

1 -> 2 -> 3 -> 4 -> 5 -> null.

## 7.4 Reverse Linked List :-

Aim :- Write a program on the head of a singly linked list, reverse the list, and return the reversed list.

PROGRAM :-

```
Public class LinkedListReverse {
  Public ListNode reverseList (ListNode head) {
    ListNode prev = null;
    ListNode current = head;
    While (current != null) {
      ListNode next = current.next;
      current.next = prev;
      Prev = current;
      current = next;
    }
    return prev;
}
Public Static void main (String[] args) {
```

```
LinkedListReverse list = new LinkedListReverse();
ListNode head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
head.next.next.next = new ListNode(4);
head.next.next.next.next = new ListNode(5);
ListNode newHead = list.reverseList(head);
ListNode temp = newHead;
while (temp != null) {
  System.out.print(temp.data + " -> ");
  temp = temp.next;
}
System.out.println("null");
}
}
class ListNode {
  int data;
  ListNode next;
  ListNode(int data) {
```

```
        this.data = data;
        this.next = null;
    }
}
```

OUTPUT :- 5 -> 4 -> 3 -> 2 -> 1 -> null.

7.5 Palindrome Linked List :-

Aim :- Write a program on the head of a singly linked list, return True if it is a Palindrome or false Otherwise.

PROGRAM :-

```
Public class PalindromeLinkedList {
  Public boolean isPalindrome (ListNode head) {
    If (head == null || head.next == null) {
      return true;
    }
    ListNode slow = head, fast = head;
    While (fast != null && fast.next != null) {
```

```
    slow = slow.next;
    fast = fast.next.next;
  }
  ListNode secondHalf = reverseList(slow);
  ListNode firstHalf = head;
  while (secondHalf != null) {
    if (firstHalf.val != secondHalf.val) {
      return false;
    }
    firstHalf = firstHalf.next;
    secondHalf = secondHalf.next;
  }
  return true;
}

public static void main(String[] args) {
  PalindromeLinkedList list = new PalindromeLinkedList();
  ListNode head = new ListNode(1);
  head.next = new ListNode(2);
  head.next.next = new ListNode(2);
```

```
        head.next.next.next = new ListNode(1);
        boolean result = list.isPalindrome(head);
        System.out.println("Is the linked list a palindrome? " +
                           result);
    }
}
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}
```

OUTPUT :-

Is the linked list a Palindrome ? True.

## 7.6 Middle Of The Linked List :-

Aim :- Write a program that returns the Middle Node of the linked list, If there are two Middle Nodes,

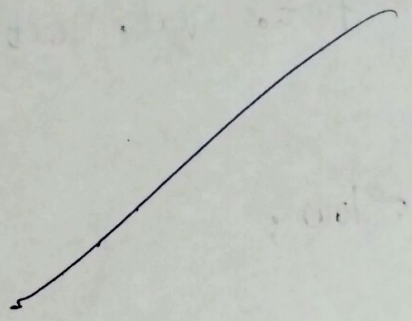
Return the Second Middle Node.

PROGRAM :-

```
Public class MiddleOfLinkedList {
  Public ListNode findMiddleNode ( ListNode head ) {
    if (head == null) {
      return null;
    }
    ListNode slow = head;
    ListNode fast = head;
    While ( fast != null && fast.next != null) {
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow;
}
Public static void main (String [] args) {
  MiddleOfLinkedList list = new MiddleOfLinkedList();
  ListNode head = new ListNode (1);
```

```
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    head.next.next.next = new ListNode(4);
    head.next.next.next.next = new ListNode(5);
    ListNode middleNode = list.findMiddleNode(head);
    System.out.println("Middle node value: " + middleNode.val);
  }
}

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



OUTPUT :-

Middle Node Value : 3.

7.7 Convert Binary Number in a Linked List to Integer :-

AIM :- Write a Program on which returns the decimal value of the number in the linked list ( The value of each node is either 0 or 1 ).

PROGRAM :-

```
Public class ConvertBinaryNumberInLinkedListToInteger {
  Public int getDecimalValue (ListNode head) {
    int decimalValue = 0;
    while (head != null) {
      decimalValue = (decimalValue << 1) | head.val;
      head = head.next;
    }
    return decimalValue;
  }
  Public Static void main (String[] args) {
    ConvertBinaryNumberInLinkedListToInteger converter =
      new ConvertBinaryNumberInLinkedListToInteger();
```

```
        ListNode head = new ListNode(1);
        head.next = new ListNode(0);
        head.next.next = new ListNode(1);
        int decimalValue = converter.getDecimalValue(head);
        System.out.println("Decimal value: " + decimalValue);
    }
}
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}
```

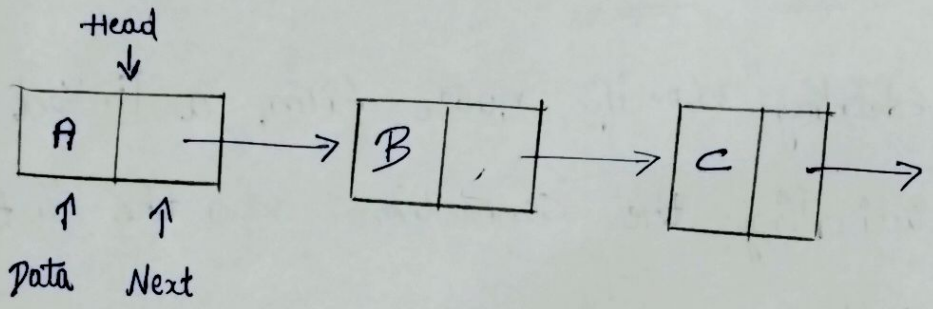
OUTPUT :-

Decimal Value : 5

## VIVA VOCE :-

1) What is Singly Linked List ?

A) A Singly Linked List is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.




2) Define Linked List Cycle ?

A) A Linked List Cycle is a condition in a linked list where a node's next pointer links back to a previous node, forming a closed loop in the list. This means there's no null end, and traversing the list could result in an

Infinite loop.

3) What is Palindrome Linked List ?

A) A palindrome linked list is a linked list that reads the same forward and backward.

(e.g. 1 → 2 → 3 → 2 → 1).

4) Define Remove & Reverse Linked List Elements ?

A) Remove Linked List Elements :-

It means deleting specific nodes from a linked list and adjusting the connections so the list remains intact.

Reverse Linked List Elements :-

It means changing the direction of the links in the lists so that the first node becomes the last and the last node becomes the first.

Kavi