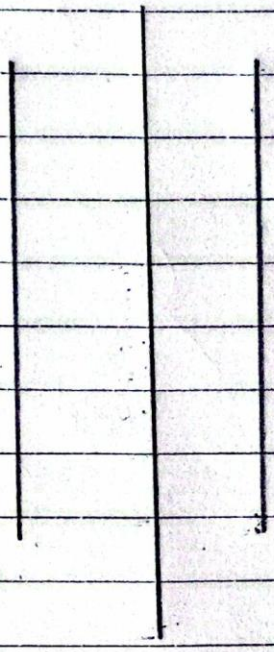
GOOD MORNING
PAGE NO. :
DATE : __/__/____

LA GRANDEE INTERNATIONAL ~~COLLEGE~~ COLLEGE



Data Structure and Algorithm.

Assignment II

Name: Sachin Timilsina
BCA 3rd Sem (2025)

Submitted To:
Er. Ashwin Poudel.

DSA.

GOOD MORNING
PAGE NO.:
DATE: / /

Assignment 2

1. WAP to implement a singly linked list.

Algorithm.

1. Create a node.
   a) Make a structure that can hold two things:
      - Data
      - Reference to next node.

2. Create a linked list.
   a) • Keep a pointer called head that point to the first node.
   b) Initially set head to empty.

3) Insert a new element at end.
   a) Create new node.
   b) Put the value inside new node.
   c) If list is empty, make the head point to this new node.
   d) Otherwise:
      - Start from head and move through the list until you reach last node.
      - Make the last node reference new node.

Time Complexity: O(n) for Insdt a new element at end.
Space Complexity: O(1) for Insert a new element at end.

```java
public class SinglyLinkedList {
    Node head;
    static class Node {
        int data;
        Node next;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // Constructor for the SinglyLinkedList.
    public SinglyLinkedList() {
        head = null;
    }

    public void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node last = head;

            while (last.next != null) {
                last = last.next;
            }
            last.next = newNode;
        }
```

```
    }

    public void display() {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        Node current = head;
        while (current != null) {
            System.out.println(current.data);
            current = current.next;
        }
    }

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

        list.insertAtEnd(10);
        list.insertAtEnd(20);
        list.display();
    }
}
```

GOOD MORNING
PAGE NO. :
DATE : __/__/____

2. WAP to implement stack using linked list.

Algorithm:

1) Make a node with (data & reference to next node).
2) Initialize stack by keeping a pointer called top, initially empty.
3) To push.
   a) Create a new node
   b) Put value inside new node
   c) Make new node's next pointer point to the current top node.
   d) Move top pointer so it points to new node.
4) To pop.
   a) If empty, "nothing to pop".
   b) Store the value of top in temp. variable.
   c) Move the top pointer to next node (top = top.next)
   d) ~~Return stored value.~~ Delete old top node.
   e) Return temp. value.
5) To peek.
   1) If top is empty, ~~repor~~ nothing to show.
   2) Return value stored on top. node.

~~Push~~	Push	Pop	Peek.
Time Complexity	O(1)	O(1)	O(1)
Space Complexity	O(1)	O(1)	O(1)

```
public class StackUsingLinkedList {
    Node top;

    static class Node {
        int data;
        Node next;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // Constructor for the Stack. Sets top to null
(empty).
    public StackUsingLinkedList() {
        top = null;
    }

    public void push(int data) {
        Node newNode = new Node(data);

        newNode.next = top;

        top = newNode;
        System.out.println(data + " pushed to stack");
    }

    public int pop() {
        if (top == null) {
```

```
            System.out.println("Stack is empty. Nothing
to pop.");
            return Integer.MIN_VALUE;
        }

        int poppedValue = top.data;
        top = top.next;
        return poppedValue;
    }

    public int peek() {
        if (top == null) {
            System.out.println("Stack is empty. Nothing
to peek.");
            return Integer.MIN_VALUE;
        }

        return top.data;
    }

    public boolean isEmpty() {
        return (top == null);
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return;
        }

        Node current = top;
        while (current != null) {
```

```
            System.out.println(current.data);
            current = current.next;
        }
    }

    public static void main(String[] args) {
        StackUsingLinkedList stack = new
StackUsingLinkedList();

        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();

        System.out.println("Top element is (peek): " +
stack.peek());

        System.out.println("Popped: " + stack.pop());
        stack.display();
    }
}
```

GOOD MORNING
PAGE NO.:
DATE: __/__/__

3. WAP to implement queue using linked list.

Algorithm:

1) Create a node with value & reference to next node.
2) Initialize a queue with two pointer front & rear to empty.
3) To enqueue:
   a) Create a new node.
   b) Put value inside new node.
   c) Set next pointer of new node to empty.
   d) If queue is empty:
      - Set both front & rear to this new node.
      - Stop.
   e) Otherwise.
      - Make current rear node next point to new node.
      - Move rear to new node.
4) To dequeue:
   a) If queue is empty, nothing to dequeue.
   b) Store the value of front node.
   c) Move front to next of it's node.
   d) After moving if front is empty, set rear to empty (queue is empty)
   e) Delete front node.
   f) Return stored value.

	Enqueue	Dequeue
Time Complexity	O(1)	O(1)
Space Complexity	O(1)	O(1)

```
public class QueueUsingLinkedList {

    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    static class Queue {
        Node front;
        Node rear;

        public Queue() {
            this.front = null;
            this.rear = null;
        }

        public void enqueue(int value) {
            Node newNode = new Node(value);

            if (front == null) {
                front = newNode;
                rear = newNode;
                // Stop
                return;
            }

            rear.next = newNode;
```

```
        rear = newNode;
    }

    public int dequeue() {
        if (front == null) {
            System.out.println("Queue is Empty");
            return -1;
        }

        int storedValue = front.data;

        front = front.next;

        if (front == null) {
            rear = null;
        }

        return storedValue;
    }

    public void display() {
        if (front == null) {
            System.out.println("Queue is Empty");
            return;
        }
        Node temp = front;
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
```

```
    }

    public static void main(String[] args) {
        Queue q = new Queue();

        q.enqueue(10);

        q.enqueue(20);

        q.enqueue(30);
        q.display();

        System.out.println("Dequeued: " + q.dequeue());
        q.display();

        System.out.println("Dequeued: " + q.dequeue());
        q.display();

        System.out.println("Dequeued: " + q.dequeue());
        q.display();

        System.out.println("Attempting to dequeue from
empty queue:");
        q.dequeue();
    }
}
```

GOOD MORNING
PAGE NO. :
DATE : __/__/____

4 WAP to insert a node in the middle of the linked list.

Given : Linked list with head pointer & a value to insert & position.

Algorithm :

1) Create a new node & store the value inside it.
2) If position is 0:
   a) Make new node point to current head.
   b) Move head pointer to new node.
   c) Stop.
3) Otherwise,
   a) Start from head.
   b) Move through the list until you reach the node just before target position.
   c) Keep counter as you move.
   d) If you run out of nodes before reaching position, invalid position.
   e) Once you reach correct previous position.
   f) Make new node next point to previous next node.
   g) Set previous node next to new node.
   h) If new node is inserted at end, update tail pointer.

Time Complexity : ~~O(1)~~ O(n)
Space Complexity : O(1).

```
public class LinkedListInsertion {

    // Node class structure
    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    static class LinkedList {
        Node head;
        Node tail;

        public void insertAtPosition(int value, int
position) {
            Node newNode = new Node(value);

            if (position == 0) {
                newNode.next = head;

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

                return;
            }
```

```
            Node current = head;
            int currentPos = 0;

            while (current != null && currentPos <
position - 1) {
                current = current.next;
                currentPos++;
            }

            if (current == null) {
                System.out.println("Error: Invalid
position " + position);
                return;
            }

            newNode.next = current.next;
            current.next = newNode;

            if (newNode.next == null) {
                tail = newNode;
            }
        }

        public void display() {
            Node temp = head;
            while (temp != null) {
                System.out.println(temp.data);
                temp = temp.next;
            }
        }
    }
```

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

        list.insertAtPosition(10, 0);
        list.display();

        list.insertAtPosition(20, 1);
        list.display();

        list.insertAtPosition(40, 2);
        list.display();

        list.insertAtPosition(30, 2);
        list.display();

        System.out.println("Inserting at invalid
position 10:");
        list.insertAtPosition(99, 10);
    }
}
```

5. WAP to insert at the end of the linked list.

Algorithm:

1) Create a node (Data & Reference).
2) Create a linked list..
    a) Keep pointer called head that points to first node.
    b) Initially set head to empty.
3) Insert a new element at end
    a) Create a new node.
    b) Put the value inside new node
    c) If list is empty, make head point to new node.
    e) Otherwise:
        - Start from head and move through the list until you reach last node.
        - Make last node reference new node

Time Complexity : O(n)
Space Complexity = O(1)

```java
public class LinkedListEndInsertion {

    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    static class LinkedList {
        Node head;

        public LinkedList() {
            this.head = null;
        }

        public void insertAtEnd(int value) {
            Node newNode = new Node(value);

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

            Node current = head;

            while (current.next != null) {
                current = current.next;
            }
```

```
            current.next = newNode;
        }

        public void display() {
            Node temp = head;
            while (temp != null) {
                System.out.println(temp.data );
                temp = temp.next;
            }
        }
    }

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


        list.insertAtEnd(10);
        list.insertAtEnd(20);
        list.display();
    }
}
```

6. WAP to delete a node from the beginning of the linked list.

Algorithm:

1) Create a node (Data & reference).
2) Create a linked list.
    a). Keep pointer called head that points to first node.
    b) Initially set head to empty.
3) Insert a new node.
    a) Create a new node.
    b) Put the value inside new node.
    c) Make head point to new node.
4) ~~Store~~ reference to head
5) ~~a)~~ Make ~~head~~ head point to head's next.
6) Delete ~~head~~ reference.

Time Complexity : O(1)
Space Complexity : O(1)

```java
public class LinkedListBegDeletion {
    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    static class LinkedList {
        Node head;

        public LinkedList() {
            this.head = null;
        }

        public void insert(int value) {
            Node newNode = new Node(value);

            if (head == null) {
                head = newNode;
            } else {
                newNode.next = head;
                head = newNode;
            }
        }

        public void deleteFromBeginning() {
            if (head == null) {
```

```
                System.out.println("List is empty,
nothing to delete.");
                return;
            }

            System.out.println("Deleting head node with
value: " + head.data);

            head = head.next;
        }

        public void display() {
            Node temp = head;
            while (temp != null) {
                System.out.println(temp.data);
                temp = temp.next;
            }
        }
    }

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

        list.insert(30);
        list.insert(20);
        list.insert(10);

        list.display();

        list.deleteFromBeginning();
        list.display();
```

```
        list.deleteFromBeginning();
        list.deleteFromBeginning();
        list.display();

        // Try to delete from empty list
        list.deleteFromBeginning();
    }
}
```

GOOD MORNING
PAGE NO.
DATE : / /

7) WAP to delete a node from the specified position of linked list.

Given: at least one ~~data~~ node in linked list, & p. head

Algorithm:

1) ~~Create a node (Data & reference).~~
2) Create a linked list.
   a) Keep pointer called head that points to first node.
   b) Initially set head to empty.
3) ~~Insert a new element at end.~~
   a)
3) ~~Crea Insert a~~

1) Input position.
2) Look for the position, if position is beyond size, invalid position.
3) Look for previous node from position.
4) ~~Set~~ previous node next to positioned node's.
4) Reference the position node.
5) Point previous node next to ~~ret~~ positioned node's next.
6) Delete the positioned node.

Time Complexity: O(n)
Space Complexity: O(1).

```java
public class LinkedListDeletionAtPosition {
    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    static class LinkedList {
        Node head;

        public void insert(int value) {
            Node newNode = new Node(value);
            if (head == null) {
                head = newNode;
                return;
            }
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }

        public void deleteAtPosition(int position) {
            if (head == null) {
                System.out.println("List is empty.");
                return;
            }
```

```
            if (position == 0) {
                System.out.println("Deleting head node:
" + head.data);
                head = head.next;
                return;
            }

            Node prev = head;
            int count = 0;

            while (prev != null && count < position - 1)
{
                prev = prev.next;
                count++;
            }

            if (prev == null || prev.next == null) {
                System.out.println("Invalid position " +
position);
                return;
            }

            Node current = prev.next;
            System.out.println("Deleting node at pos " +
position + " with value: " + current.data);

            prev.next = current.next;
        }

        public void display() {
            Node temp = head;
```

```
            while (temp != null) {
                System.out.println(temp.data);
                temp = temp.next;
            }
        }
    }

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

        list.insert(10);
        list.insert(20);
        list.insert(30);
        list.insert(40);
        list.display();

        list.deleteAtPosition(2);
        list.display();

        list.deleteAtPosition(0);
        list.display();

        // Invalid position
        list.deleteAtPosition(10);
    }
}
```

Github Link: https://github.com/Sachin-Timilsina/DSA-Assignments