Department of Software Engineering
Mehran University of Engineering and Technology, Jamshoro

Course: SWE224 – Data Structure & Algorithms			
Instructor	Mariam Memon	**Practical/Lab No.**	06
Date		**CLOs**	3
Signature		**Assessment Score**	01

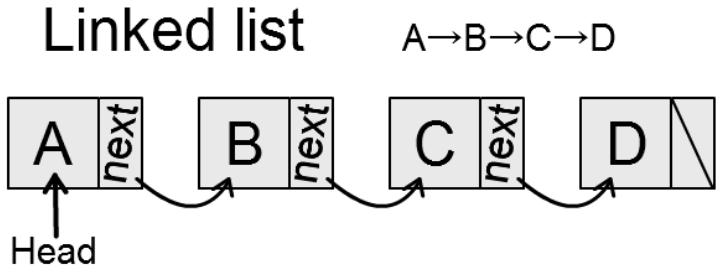
Topic	**Implementation of Linked List**
Objectives	To implement linked data structure and its various methods in Java.

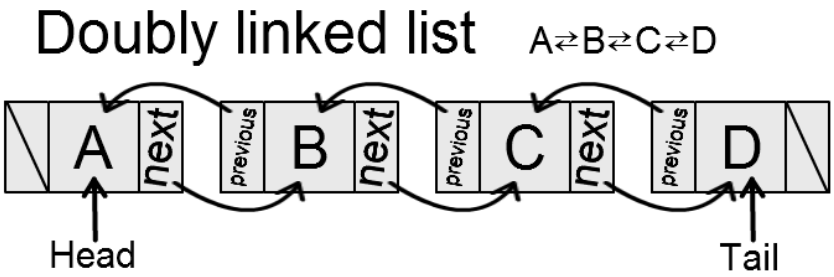
Lab Discussion: Theoretical concepts and Procedural steps

Linked List - ADT

Linked List is an *Abstract Data Type (ADT)* that holds a collection of **Nodes,** the nodes can be accessed in a sequential way. **Linked List doesn't provide a random access to a Node.**

Usually, those Nodes are connected to the next node and/or with the previous one, this gives the linked effect. When the Nodes are connected with only the next pointer the list is called Singly Linked List and when it's connected by the next and previous the list is called Doubly Linked List.





The Nodes in a Linked List can be anything from primitives types such as integers to more complex types like instances of classes.

Linked List Implementation in Java

```
class LinkedList
{
	private Node start;

	private static class Node
	{
		private int data;
		private Node next;

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

		public Node(int data, Node next)
		{
			this.data=data;
			this.next=next;
		}
	}
	//Traversing a linked list
	public void printList()
	{
		for (Node p = start; p!=null; p=p.next)
		{
			System.out.println(p.data);
		}
	}

	//Inserting element in a linked list
	public Node insert(int x)
	{
		Node p = start;
		if(start == null || start.data > x)
		{
			start = new Node(x, start);
			return start;
		}
		while(p.next != null)
		{
```

```
                if(p.next.data>x) break;
                p=p.next;
            }
            p.next = new Node(x, p.next);
            return start;
        }

        //deleting an element from a linked list
        public Node delete(int x)
        {
            if(start == null || start.data > x)
            {
                //x is not in the list
                return start;
            }
            if(start.data==x)
            {
                //x is the first element in the list
                start = start.next;
                return start;
            }
            for(Node p=start; p.next!=null; p = p.next)
            {
                if(p.next.data>x) break;
                if(p.next.data == x)
                {
                    p.next=p.next.next;
                    break;
                }
            }
            return start;
        }
}
```

Lab Tasks

Task 1: Write and test the following methods:

1. int search(int x) //returns the index of element in the linked list
2. int size() //returns the size of the linked list
3. int sum() //returns the sum of all numbers in the linked list
4. void deleteLast() //deletes the last node in the list
5. LinkedList copy() //returns a new linked list that is the duplicate of the list this method is called up on.

6. LinkedList subList(int p, int q) //returns a new linked list that contains element from node **p** to node **q** of the list this method is called up on.
7. void append(LinkedList l) //l is appended to the list this method is called on.
8. LinkedList merged(Linked l) // returns a new list that merges l with the list the method is called on; maintaining the ascending order.

Task 2: Implement a linked list for Student class. Student class should have roll_num and name as instance variables. The linked list should have the following operations:

- insert(Student s)
- delete(Student s)
- printList() // print the roll numbers and names of every student in the list

Task 3: Explore java.util.LinkedList class; Create a linked list of type String using this class and apply any 5 of its methods.

Lab Rubrics for Evaluation

Rubrics	**Proficient**	**Adequate**	**Poor**
Programming Algorithms	Completely accurate and efficient design of algorithms **(0.4)**	Accurate but inefficient algorithms **(0.2)**	Unable to design algorithms for the given problems **(0.0)**
Originality	Submitted work shows large amount of original thought **(0.3)**	Submitted work shows some amount of original thought **(0.2)**	Submitted work shows no original thought **(0.0)**
Troubleshooting	Easily traced and corrected faults **(0.3)**	Traced and corrected faults with some guidance **(0.2)**	No troubleshooting skills **(0.0)**