# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction to LinkedList

## 1.1 Introduction

- Link list is a linear list of linked elements. Like arrays, linked list represents another linear data structure.
- In arrays, there is always a fixed relationship between the addresses of two consecutive elements as all the items of an array must be stored contiguously.
- However, note that this contiguity requirement makes expansion or contraction of the size of the array difficult.
- In linked list, data items may be scattered arbitrarily all over the memory, but we can achieve fast insertion and deletion in a dynamic situation.

**Limitations of Array** In many applications, the array is not suitable as it has some drawback. The drawbacks of Array are listed below:

- The maximum size of the array needs to be predicted beforehand. One cannot change the size of the array after allocating memory, but, many applications require resizing.
- Most of the space in the array is wasted when programmer allocates arrays with large size. On the other hand, when program ever needs to process more than the specify size then the code breaks.
- Storage of the array must be available contiguously. Required storage not always immediately available.
- Insertion and deletion operation may be very slow. The worst case occurs when the first element is to be deleted or inserted. Almost all the elements of the array need to be moved.
- Joining and splitting of two or more arrays is difficult.

## 1.2 LinkedList

**Definition:** A linked list is a linear ordered collection of finite homogeneous data elements called node, where the linear order is maintained by means of links or pointers.

- A linked list allocates space for each element separately in its own block of memory called a "linked list element" or "node".
- A linked list, in its simplest form, is a collection of nodes that collectively form a linear sequence.
- The list structure is created by the use node object and connect all its nodes together like the links in a chain.

| Property | Array | Linked |
|---|---|---|
| Storage | Storage of the array must be available contiguously. | Storage need not be contiguous. |
| Memory utilization | The size of the array needs to be predicted beforehand because memory allocation is done in advance.<br><br>Not necessary for storing addresses of any element. | Memory of linked list is not pre-allocated, memory is allocated whenever it is required.<br><br>Extra memory space is necessary for storing addresses of the next node |
| Change of size | The array size is fixed, extend or shrink not possible during the execution of a program | Linked list may extend or shrink during the execution of a program |
| Insertion/ deletion | Insertion/deletion operations are slow, half of the elements are required to move on an average | Insertion/deletion operations are performed very fast, in a constant amount of time |
| Searching | Linear searching, binary searching, interpolation searching are possible | Binary searching, interpolation searching not possible, only linear searching is possible |
| Access element | Fast access to any element in a constant amount of time. | To access any element in a linked list, traversing is required. |
| Joining/ splitting | Joining and splitting of the two arrays is difficult. | Joining and splitting of two linked list is very easy. |

### 1.2.1 Advantages of linked list

Linked lists have many advantages. Some of the very important advantages are:

1. Linked list are dynamic data structures. That is, they can extend or shrink during the execution of a program.

2. Storage need not be contiguous.
3. Efficient memory utilization. Here memory is not pre-allocated. Memory is allocated whenever it is required.
4. Insertion or deletion is easy and efficient, may be done very fast, in a constant amount of times, ndependent of the size of the list.
5. The joining of two linked lists can be done by assigning pointer of the second linked list in the last node of the first linked list.

### 1.2.2 Types of Linked list

There are different types of linked list. We can put linked lists into the following four types:

1. Singly linked list
2. Circular Linked list
3. Doubly Linked list
4. Circular Doubly linked list

## 1.3 Singly linked list

An element in a linked list is specially termed as a node. In a singly linked list, each node consists of two fields:

1. **data** field that contains the actual information of the element.
2. **link** or **next** field, contains the address of the next node in the list.
3. **head**, the linked list instance must keep a reference to the first node of the list, known as the head.


Figure 1.1: Single node

- A "data" field to store whatever element type the list holds for the user, and a "link or next" field, which is a pointer, used to link one node to the next node.
- **head:** The linked list instance must keep a reference to the first node of the list, known as the **head**.

- **tail:** The last node of the list is known as the **tail**.
- The tail of a list can be found by traversing the linked list— starting at the head and moving from one node to another by following each node's next reference.
- We can identify the tail as the node having **null** as its next reference. This process is also known as **link hopping** or **pointer hopping**.


Figure 1.2: Single LinkedList

## 1.4 Operations on Singly linked list

Operations supported by a singly linked list are as follows:

- **Createlist:** This operation creates a linked list.
- **Traverse:** This operation traverse/visit all the elements of the linked list exactly once
- **Insertion:** This operation inserts an element to the linked list
- **Deletion:** This operation removes an element from the linked list
- **Searching:** This operation performs linear searching for a key value in the linked list
- **Reverse:** This operation performs the reverse of the linked list
- **Merging:** This operation performs merging of two linked lists in a single linked list

### 1.4.1 Create List

The following algorithm creates a node and appends it at the end of the existing list.

- *'Head'* is a pointer which holds the address of the HEADER of the linked list.
- *'item'* is the data or value of the new node.
- *next* is a field which holds the address of the new node
- 'Temp' is a temporary node.

**Algorithm 1.1 create (Head, item)**

```
    [Create new Node]
 1: Create the new Node
 2: Set Node.data = item
 3: Set Node.next = NULL
    [Check for empty List]
 4: if Head = NULL then
        Set Head = Node
 5: else
        Set Temp = Head
 6:     while Temp.next ≠ null do
            Set Temp = Temp.next
 7:     end while
 8:     Temp.next = Node
 9: end if
10: Return Head
```

- In the abov ealgorithm **Node** represent a new node of the Linked list.
- At first creates a **Node** object and assign memory**[Statement-1]**.
- Assign the value or **item** to the **data** part of the **Node [Statement-2]**
- Store NULL in the **next** part of the **Node**. **[Statement-3]**.

#### 1.4.1.1 Java Program to Create a new Node and Linked List

In java we can created a class named Node to represent a node object. The class has two members -

- the first is the **data** of integer type.It can be any data value that we want to store in the Node object.
- the other is the Node class type reference **next** which referes to the memory of the next node in the linked list.
- So when we represent any node for the linked list, it refers to an object of **Node class** type.

```
class Node {
   public int data;
   public Node next;
}
```

The Node class is used in the **createNode(Node head, int item)** method. It can be called repeatedly for creating the nodes.

- The **node,temp, head** are are Node class objects.

```
public static Node create(Node head, int item) {
   Node node=new Node();
   Node temp=new Node();
   node.data=item;
   node.next=null;
   if(head==null)
   head=node;
   else {
      temp=head;
      while(temp.next != null) {
         temp=temp.next;
      }
      Temp.next=node;
   }
   return head;
}
```

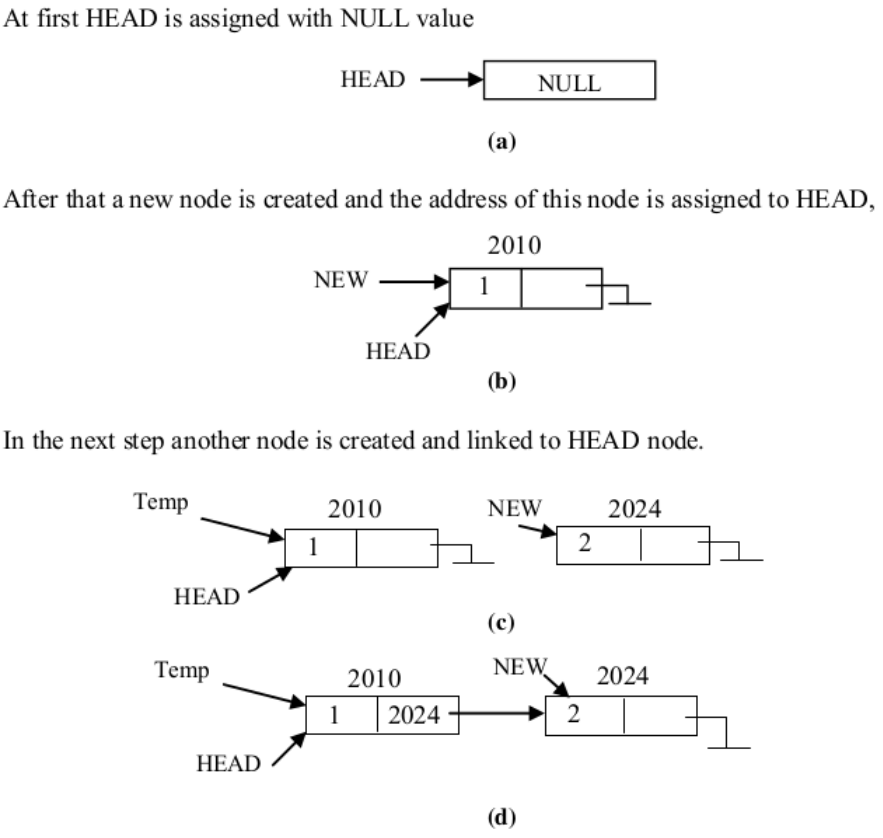


Figure 1.3: Create Single LinkedList

### 1.4.2 Traversing / Display a Linked List

This algorithm traverses a linked list and prints the data part of each node of the linked list.

- The '_HEAD' is a pointer which points to the starting node of the linked list
- '_Temp' is a temporary pointer to traverse the list.

**Algorithm 1.2 Traverse (Head)**

1: **if** Head = NULL **then**
   Print: "The Linked list is empty"
   Return Head
2: **end if**
3: Temp = Head
4: **while** $Temp \neq NULL$ **do**
   a) print: Temp.data
   b) Set Temp = Temp.next
5: **end while**
6: Return Head

**Java method to traverse a LinkedList**

```
public static Node traverse(Node head) {
   Node temp;
   if(head==null) {
      System.out.println("Empty LinkedList");
      return head;
   }
   else {
      System.out.println("Data of the Linked List:");
      Temp=head;
      while(Temp != null) {
         System.out.print(Temp.data+" ");
         Temp=temp.next;
      }
   }
   return head;
}
```

**Complete Java Program to Create and Traverse a Linked List**

Compete java code to create and Traverse a single LinkedList. It use a Node class for the node, two methods to create and display nodes of the LinkedList

```
package insertnode;
import java.util.Scanner;
//Class for the contents of a node
class Node{
   int data;
   Node next;
}
class LinkedList{
   //Method t create a new node by using the Node class object
   public static Node create(Node head) {
      Scanner sc=new Scanner(System.in);
      char choice;
      int nodeCount=1;
      do {
         Node node=new Node();
         Node temp;
         System.out.println("Enter data for the new node");
         node.data=sc.nextInt();
         node.next=null;
         if(head==null)
            head=node;
         else {
            temp=head;
            while(temp.next != null) {
               temp=temp.next;
            }
            temp.next=node;
         }
         System.out.println(nodeCount+" node created");
         System.out.println("Would you like to enter another node(y|n)");
         choice=sc.next().charAt(0);
         if(choice=='y' || choice=='Y')
         nodeCount++;
      }while(choice=='y' || choice=='Y');
      return head;
   }
   //Method to display the linked list
   public static void display(Node head) {
      Node temp;
```

```
      temp=head;
      if(head==null)
      System.out.println("Empty linked list");
      else {
         System.out.println("The elements of the linked list are:");
         while(temp != null) {
            System.out.print(temp.data+"-->");
            temp=temp.next;
         }
      }
   }
}
public class LinkedListInsertDriver {
   public static void main(String[] args) {
      Node head=null;
      head=LinkedList.create(head);
      LinkedList.display(head);
      head=LinkedList.insertBegin(head);
      LinkedList.display(head);
   }
}
```

**Java program to create node and assign value using constructor. Link the nodes manually and use different methods to traverse the node of the linked list**

```
package simplecreate;
class Node{
   int data;
   Node next;
   Node(){}
   Node(int data){
      this.data=data;
   }
}
public class DisplayLinkedList {
   //Method to find the length of the linked list
   public static int length(Node head) {
      int count=0;
      while(head!=null) {
         count++;
         head=head.next;
      }
      return count;
   }
   //Method to display the linked list
   public static void display(Node head) {
      Node temp=head;
      if(head==null)
         return;
      while(temp!=null) {
         System.out.print(temp.data+"-->");
         temp=temp.next;
      }
   }
   //Method for the recursive display of the linked list
   public static void recDisplay(Node head) {
      if(head==null)
         return;
      System.out.print(head.data+"-->");
      recDisplay(head.next);
   }
   //Method to reverse display of the linked list using recursive display
   public static void reverseDisplay(Node head) {
      if(head==null)
         return;
```

```
      reverseDisplay(head.next);
      System.out.print(head.data+"-->");
   }
public static void main(String[] args) {
   Node n1=new Node(5);
   Node n2=new Node(3);
   Node n3=new Node(2);
   Node n4=new Node(7);
   Node n5=new Node(10);

   //Connect the nodes
   n1.next=n2;
   n2.next=n3;
   n3.next=n4;
   n4.next=n5;

   //Find length of the lined list
   System.out.println("Length of the linked list = "+length(n1));

   //Display the Linked list using for loop
   System.out.println("\nDisplay the LinkedList using for loop ");
   Node temp=n1;
   for(int i=0;i<5;i++) {
      System.out.print(temp.data+"-->");
      temp=temp.next;
   }

   //Display the Linked list using display method
   System.out.println("\nDisplay the Linked list using display method");
   display(n1);

   //Display the Linked list using recurssion
   System.out.println("\nDisplay the Linked list using recurssion");
   recDisplay(n1);

   System.out.println("\nReverse Display the Linked list using recurssion");
   reverseDisplay(n1);
   }

}
```

### 1.4.3 Insertion in Singly Linked List

Insertion operation in a singly linked list can be done in different ways using position.

- Insertion at beginning.
- Insertion at end.
- Insertion at any index.

#### 1.4.3.1 Insertion of a node at first position of a singly linked list

In the following algorithm insertion of node at beginning position is described. The '_HEAD' is a pointer which points to the starting node of the linked list. Node points to the new node.

**Algorithm 1.3 insertAtBegin (Head, item)**

**[Create new Node]**
1: Allocate memory for the new Node node
2: Set Node.data = item
3: Set Node.next = Head
**[Make the HEADER to point to the new Node]**
4: Set Head=Node
5: Return Head

The following diagrams explain the insertion operation at the beginning of a singly linked list. At first HEAD points to the first node of the list containing two nodes.

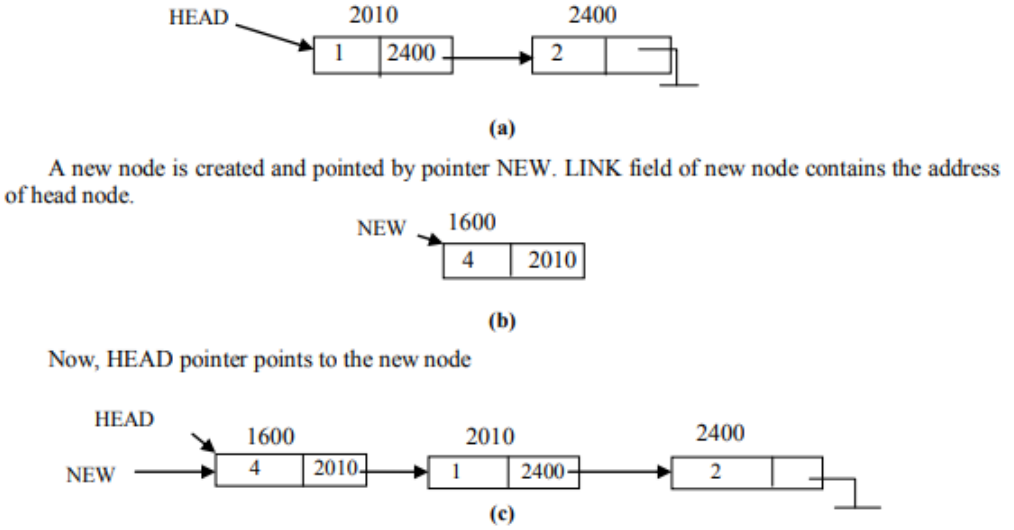


Figure 1.4: Node insertion in a single linked list at beginning

**Java method to insert a node at the beginning of a linked list**

```
public static Node insertAtBeg(Node head) {
   Scanner sc=new Scanner(System.in);
   Node node=new Node();
   System.out.println("Enter data for new node:");
   node.data=sc.nextInt();
   node.next=head;
   head=node;
   return head;
}
```

#### 1.4.3.2 Insertion of a node at the end of a singly linked list

**Algorithm 1.4 insertAtEnd (Head, item)**

**[Create new Node]**

1: Allocate memory for the new Node node
2: Set Node.data = item
3: Set Node.next = NULL
4: Set Temp=Head [to make Temp to point to the first node]
5: **while** $Temp.next \neq null$ **do**
Set Temp = Temp.next
6: **end while**
7: Set Temp.next=Node
8: Return Head

**Java method to insert a new node at the end of a linked list**

```
public static Node insertAtEnd(Node head) {
   Scanner sc=new Scanner(System.in);
   Node node=new Node();
   System.out.println("Enter data for new node:");
   node.data=sc.nextInt();
   node.next=null;
   Node temp=head;
   while(temp.next != null) {
      temp=temp.next;
   }
   temp.next=node;
   return head;
}
```

#### 1.4.3.3 Insertion of a node at any index position of a single linked list

```
public static int size(Node head) {
   Node temp=head;
   int count=0;
   while(temp != null) {
      count++;
      temp=temp.next;
   }
   return count;
}
public static Node insertAtIndex(Node head) {
   Scanner sc=new Scanner(System.in);
   Node temp=head;
   Node node=new Node();

   //Create new node and assign required data
   System.out.println("\nEnter data for new node:");
   node.data=sc.next();
   int length=size(head);

   //Specify the index position to insert the new node
   System.out.println("Enter the position to insert the new node");
   int indx=sc.nextInt();
   //If the index is at starting
   if(indx==0)
      head=insertAtBeg(head);
   //If the index is at the end of the linked list
   else if(indx==length)
      head=insertAtEnd(head);
   //At any place except start and end of the list
   else {
      for(int i=0;i<indx-1;i++) {
         temp=temp.next;
      }
      node.next=temp.next;
      temp.next=node;
   }
   return head;
}
```

**Complete java program to perform node creation and inserction operation in a single LinkedList**

```
package insertany;
import java.util.Scanner;
class Node{
   int regdNo;
   float mark;
   Node next;
}
public class LinkedList {
   //Method to create a Linked List
   public static Node createNode(Node head) {
      Scanner sc=new Scanner(System.in);
      char choice;
      do {
         Node node=new Node();
         Node temp;
         System.out.println("Enter Regd.No.");
         node.regdNo=sc.nextInt();
         System.out.println("Enter mark:");
         node.mark=sc.nextFloat();
         if(head==null)
            head=node;
         else {
            temp=head;
            while(temp.next != null) {
               temp=temp.next;
            }
            temp.next=node;
         }
         System.out.println("Want to create one more node(y/n) ");
         choice=sc.next().charAt(0);
      }while(choice=='y' || choice=='Y');
      return head;
   }
   //Method to compute the size of the linked list
   public static int size(Node head) {
      Node temp=head;
      int count=0;
      while(temp != null) {
      count++;
      temp=temp.next;
```

```
    }
    return count;
  }
  //Method to display the Linked List
  public static void display(Node head) {
    Node temp=head;
    if(head==null) {
    System.out.println("Empty linked list");
    return;
    }
    while(temp != null) {
      System.out.print("("+temp.regdNo+", "+temp.mark+")-->");
      temp=temp.next;
    }
  }
  //Method to insert a new node at begining of the linked list
  public static Node insertAtBeg(Node head) {
    Scanner sc=new Scanner(System.in);
    Node node=new Node();
    System.out.println("Enter mark for new node:");
    node.mark=sc.nextFloat();
    System.out.println("Enter registration number for new node");
    node.regdNo=sc.nextInt();
    node.next=head;
    head=node;
    return head;
  }
  //Method to insert a new node at end of the linked list
  public static Node insertAtEnd(Node head) {
    Scanner sc=new Scanner(System.in);
    Node node=new Node();
    System.out.println("Enter mark for new node:");
    node.mark=sc.nextFloat();
    System.out.println("Enter registration number for new node");
    node.regdNo=sc.nextInt();
    node.next=null;
    Node temp=head;
    while(temp.next != null) {
      temp=temp.next;
    }
    temp.next=node;
    return head;
  }
```

```
   //Method to insert a new node at any index position of the linked list
   public static Node insertAtIndex(Node head) {
      Scanner sc=new Scanner(System.in);
      Node node=new Node();
      Node temp=head;
      System.out.println("\nEnter mark for new node:");
      node.mark=sc.nextFloat();
      System.out.println("Enter registration number for new node:");
      node.regdNo=sc.nextInt();
      int length=size(head);

      System.out.println("Enter the position to insert the new node");
      int indx=sc.nextInt();
      if(indx==0)
         head=insertAtBeg(head);
      else if(indx==length)
         head=insertAtEnd(head);
      else {
         for(int i=0;i<indx-1;i++) {
            temp=temp.next;
         }
         node.next=temp.next;
         temp.next=node;
      }
      return head;
   }
   public static void main(String[] args) {
      Node start=null;
      start=createNode(start);
      System.out.println("Size of linked list: "+size(start));
      display(start);
      start=insertAtIndex(start);
      display(start);
   }

}
```

### 1.4.4 Delete Node from Single Linked List

Deletion operation in a singly linked list can be done in different ways using position.

- Deletion from beginning.
- Deletion from end.
- Deletion in the middle.

#### 1.4.4.1 Delete node from begining

```
public static Node deleteBegin(Node head) {
   if(head==null) {
      System.out.println("Empty list");
      return head;
   }
   System.out.println("Deleting: "+head.data);
   head=head.next;
   return head;
}
```

#### 1.4.4.2 Delete from the End

**Method-1**

```
public static Node deleteEnd(Node head) {
   int length=size(head);
   Node temp=head;
   for(int i=0;i<length-2;i++) {
      temp=temp.next;
   }
   temp.next=temp.next.next;
   return head;
}
```

**Method-2**

```
public static Node deleteEnd2(Node head) {
   Node temp=head;
   Node prev=null;
   while(temp.next != null) {
      prev=temp;
      temp=temp.next;
   }
```

```
   prev.next=null;
   return head;
}
```

#### 1.4.4.3 Reverse a Linked List

```
public static Node reverse(Node head) {
   Node prev=new Node();
   Node temp=new Node();
   Node current=new Node();
   current=head;
   prev=null;
   while(current != null) {
      temp=current.next;
      current.next=prev;
      prev=current;
      current=temp;
   }
   head=prev;
   return head;
}
```

**Appendix Title Here**

Write your Appendix content here.