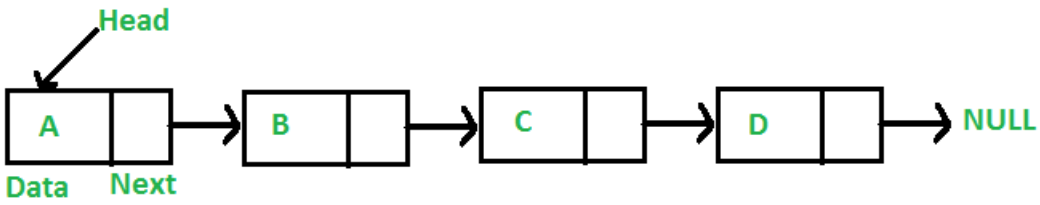
Linked List | Introduction

> *Linked Lists are linear or sequential data structures in which elements are stored at non-contiguous memory locations and are linked to each other using pointers.*

Like arrays, linked lists are also linear data structures but in linked lists elements are not stored at contiguous memory locations. They can be stored anywhere in the memory but for sequential access, the nodes are linked to each other using pointers.

Each element in a linked list consists of two parts:

- **Data**: This part stores the data value of the node. That is the information to be stored at the current node.
- **Next Pointer**: This is the pointer variable or any other variable which stores the address of the next node in the memory.



Advantages of Linked Lists over Arrays: Arrays can be used to store linear data of similar types, but arrays have the following limitations:

1. The size of the arrays is fixed, so we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the

dynamic and the size of the linked list can be incremented or decremented

have to shift. For example, in a system, if we maintain a sorted list of IDs in an array id[].

```
id[] = [1000, 1010, 1050, 2000, 2040].
```

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000). Deletion is also expensive with arrays unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved. On the other hand, nodes in linked lists can be inserted or deleted without any shift operation and is efficient than that of arrays.

Disadvantages of Linked Lists:

1. Random access is not allowed in Linked Lists. We have to access elements sequentially starting from the first node. So, we cannot do a binary search with linked lists efficiently with its default implementation. Therefore, lookup or search operation is costly in linked lists in comparison to arrays.
2. Extra memory space for a pointer is required with each element of the list.
3. Not cache-friendly. Since array elements are present at contiguous locations, there is a locality of reference which is not there in the case of linked lists.

Representation of Linked Lists

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head node of the list. If the linked list is empty, then the value of the head node is NULL.

Each node in a list consists of at least two parts:

1. data
2. Pointer (Or Reference) to the next node

In C/C++, we can represent a node using structure. Below is an example of a linked

```
{
    int data;
    struct Node* next;
```

In Java, LinkedList can be represented as a class, and the Node as a separate class. The LinkedList class contains a reference of the Node class type.

```
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;

        // Constructor to create a new node
        // Next is by default initialized
        // as null
        Node(int d) {data = d;}
    }
}
```

Below is a sample program in both C/C++ and Java to create a simple linked list with 3 Nodes.

C++ Java

```
// A simple Java program to introduce a linked list
class LinkedList
{
```

```
    /* Linked list Node. This inner class is made static so that
        Node next;
        Node(int d) { data = d; next=null; } // Constructor
    }

    /* method to create a simple linked list with 3 nodes*/
    public static void main(String[] args)
    {
        /* Start with the empty list. */
        LinkedList llist = new LinkedList();

        llist.head = new Node(1);
        Node second = new Node(2);
        Node third = new Node(3);

        /* Three nodes have been allocated dynamically.
        We have refernces to these three blocks as first,
        second and third

        llist.head     second             third
            |            |                |
            |            |                |
        +----+------+     +----+------+     +----+------+
        | 1 | null |     | 2 | null |     | 3 | null |
        +----+------+     +----+------+     +----+------+ */

        llist.head.next = second; // Link first node with the second node

        /* Now next of first Node refers to second. So they
            both are linked.

        llist.head     second             third
            |            |                |
            |            |                |
        +----+------+     +----+------+     +----+------+
        | 1 | o-------->| 2 | null |     | 3 | null |
        +----+------+     +----+------+     +----+------+ */
```

```
        /* Now next of second Node refers to third. So all three
```



```
               |              |                  |
               |              |                  |
        +----+------+     +----+------+     +----+------+
        | 1  | o-------->| 2  | o-------->| 3  | null |
        +----+------+     +----+------+     +----+------+ */
    }
}
```

Linked List Traversal: In the previous program, we have created a simple linked list with three nodes. Let us traverse the created list and print the data of each node. For traversal, let us write a general purpose function printList() that prints any given list.

C++ Java

```
// A simple Java program for traversal
// of a linked list

class LinkedList
{
    Node head; // head of list

    /* Linked list Node. This inner class is made static so that
       main() can access it */
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; next=null; } // Constructor
    }

    /* This function prints contents of linked
           list starting from head */
    public void printList()
    {
```

```
        {

    }

    /* method to create a simple linked list with 3 nodes*/
    public static void main(String[] args)
    {
        /* Start with the empty list. */
        LinkedList llist = new LinkedList();

        llist.head     = new Node(1);
        Node second     = new Node(2);
        Node third     = new Node(3);

        llist.head.next = second; // Link first node with the second node
        second.next = third; // Link first node with the second node

        llist.printList();
    }
}
```

Output:

```
1 2 3
```