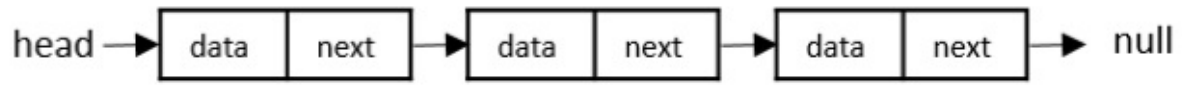
## 1.3. LINKED LISTS – SINGLE, DOUBLE, CIRCULAR LINKED LIST

A linked list is a linear data structure which can store a collection of "nodes" connected together via links i.e. pointers. Linked lists nodes are not stored at a contiguous location, rather they are linked using pointers to the different memory locations. A node consists of the data value and a pointer to the address of the next node within the linked list.

A linked list is a dynamic linear data structure whose memory size can be allocated or de-allocated at run time based on the operation insertion or deletion, this helps in using system memory efficiently. Linked lists can be used to implment various data structures like a stack, queue, graph, hash maps, etc.




A linked list starts with a **head** node which points to the first node. Every node consists of data which holds the actual data (value) associated with the node and a next pointer which holds the memory address of the next node in the linked list. The last node is called the tail node in the list which points to **null** indicating the end of the list.

### Linked Lists vs Arrays

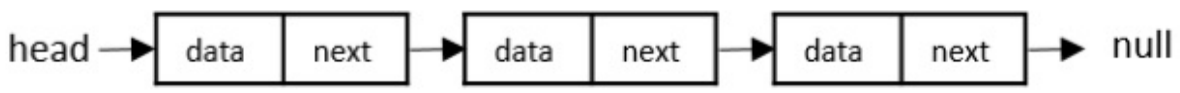
In case of arrays, the size is given at the time of creation and so arrays are of fixed lenghth where as Linked lists are dynamic in size and any number of nodes can be added in the linked lists dynamically. An array can accommodate similar types of data types where as linked lists can store various nodes of different data types.

### Types of Linked List

Following are the various types of linked list.

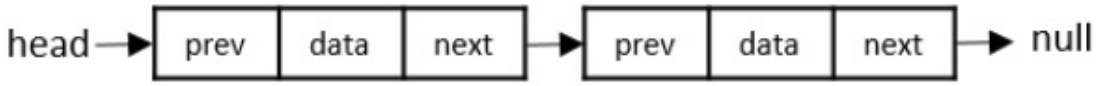
### Singly Linked Lists

Singly linked lists contain two "buckets" in one node; one bucket holds the data and the other bucket holds the address of the next node of the list. Traversals can be done in one direction only as there is only a single link between two nodes of the same list.



## Doubly Linked Lists

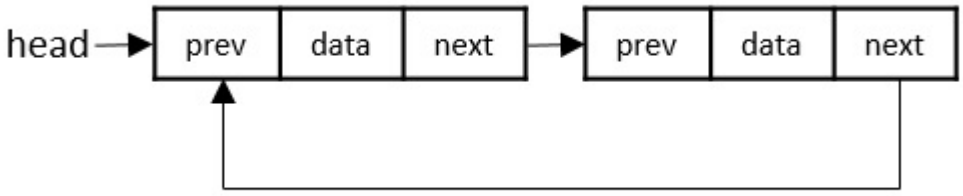
Doubly Linked Lists contain three "buckets" in one node; one bucket holds the data and the other buckets hold the addresses of the previous and next nodes in the list. The list is traversed twice as the nodes in the list are connected to each other from both sides.




## Circular Linked Lists

Circular linked lists can exist in both singly linked list and doubly linked list.

Since the last node and the first node of the circular linked list are connected, the traversal in this linked list will go on forever until it is broken.



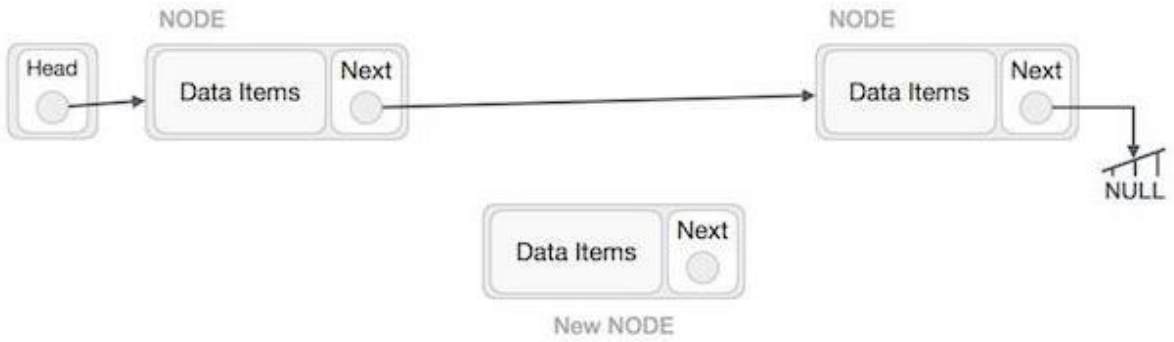

### Basic Operations in Linked List

The basic operations in the linked lists are insertion, deletion, searching, display, and deleting an element at a given key. These operations are performed on Singly Linked Lists as given below −

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

## Linked List - Insertion Operation

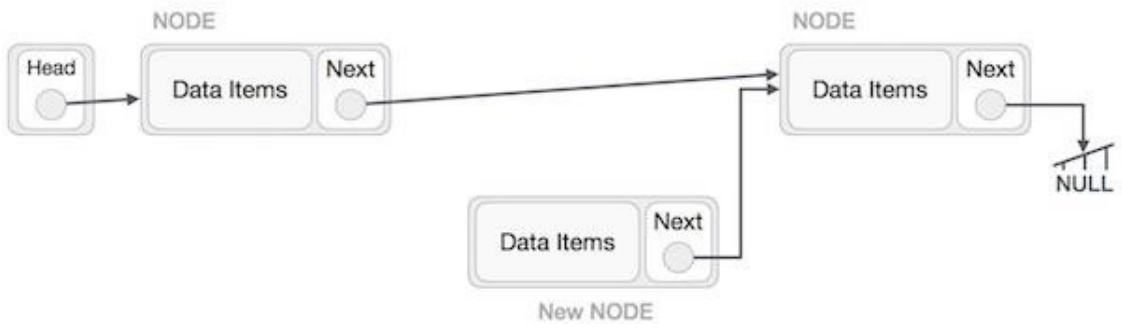
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.




Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B.next to C –

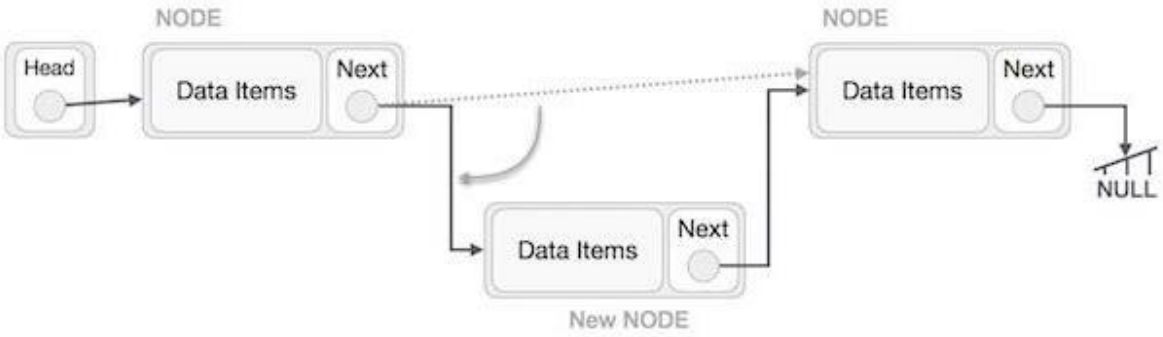
```
NewNode.next -> RightNode;
```

It should look like this –




```
LeftNode.next -> NewNode;
```

This will put the new node in the middle of the two. The new list should look like this –




Insertion in linked list can be done in three different ways. They are explained as follows −

Insertion at Beginning

In this operation, we are adding an element at the beginning of the list.

ALGORITHM

```
1. START
2. Create a node to store the data
3. Check if the list is empty
4. If the list is empty, add the data to the node and
   assign the head pointer to it.
5. If the list is not empty, add the data to a node and link to the
   current head. Assign the head to the newly added node.
6. END
```

**Applications of Linked Lists:**

- Linked Lists can be used to implement stacks, queue, deque, sparse matrices and adjacency list representation of graphs.
- Dynamic memory allocation in operating systems and compilers (linked list of free blocks).
- Manipulation of polynomials
- Arithmetic operations on long integers.
- In operating systems, they can be used in Memory management, process scheduling (for example circular linked list for round robin scheduling) and file system.
- Algorithms that need to frequently insert or delete items from large collections of data.
- LRU cache, which uses a doubly linked list to keep track of the most recently used items in a cache.

**Applications of Linked Lists in real world:**

- The list of songs in the music player are linked to the previous and next songs.
- In a web browser, previous and next web page URLs can be linked through the previous and next buttons (Doubly Linked List)
- In image viewer, the previous and next images can be linked with the help of the previous and next buttons (Doubly Linked List)
- Circular Linked Lists can be used to implement things in round manner where we go to every element one by one.
- Linked List are preferred over arrays for implementations of Queue and Deque data structures because of fast deletions (or insertions) from the front of the linked lists.

**Disadvantages of Linked Lists:**

Linked lists are a popular data structure in computer science, but like any other data structure, they have certain disadvantages as well. Some of the key disadvantages of linked lists are:

- **Slow Access Time:** Accessing elements in a linked list can be slow, as you need to traverse the linked list to find the element you are looking for, which is an O(n) operation. This makes linked lists a poor choice for situations where you need to access elements quickly.
- **Pointers or References:** Linked lists use pointers or references to access the next node, which can make them more complex to understand and use compared to arrays. This complexity can make linked lists more difficult to debug and maintain.
- **Higher overhead:** Linked lists have a higher overhead compared to arrays, as each node in a linked list requires extra memory to store the reference to the next node.
- **Cache Inefficiency:** Linked lists are cache-inefficient because the memory is not contiguous. This means that when you traverse a linked list, you are not likely to get the data you need in the cache, leading to cache misses and slow performance.