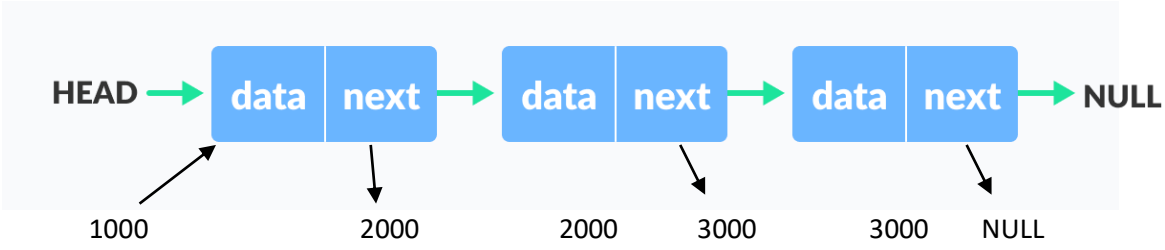
LINKED LIST

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
- The elements in a linked list are linked using pointers.
- In simple words, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.
- A linked list data structure includes a series of connected nodes.

- You have to start somewhere, so we give the address of the first node a special name called HEAD.
- Also, the last node in the linked list can be identified because its next portion points to NULL.
- For example,



Representation of Linked List

- Each node consists:

I. **A data item**
II. **An address of another node**

- We wrap both the data item and the next node reference in a struct as:
- Each struct node has a data item and a pointer to another struct node.
- Create a simple Linked List with three items

- The power of Linked List comes from the ability to break the chain and rejoin it.
- E.g. if you wanted to put an element 4 between 1 and 2, the steps would be:

1. **Create a new struct node and allocate memory to it.**
2. **Add its data value as 4**
3. **Point its next pointer to the struct node containing 2 as the data value**
4. **Change the next pointer of "1" to the node we just created.**

TYPES OF LINKED LIST

- There are three common types of Linked List.

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List

- It is the most common.
- Each node has data and a pointer to the next node.

Doubly Linked List

- We add a pointer to the previous node in a doubly-linked list.
- Thus, we can go in either direction: forward or backward.

Circular Linked List

- A circular linked list is a variation of a linked list in which the last element is linked to the first element.
- This forms a circular loop.

A circular linked list can be either singly linked or doubly linked.

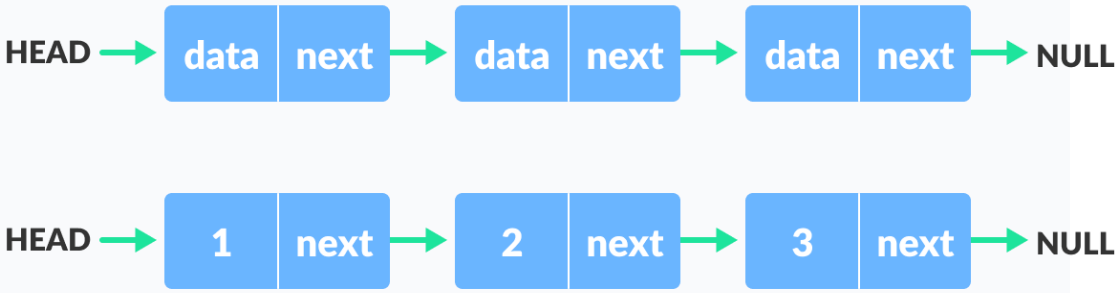
- For singly linked list, next pointer of last item points to the first item
- In the doubly linked list, prev pointer of the first item points to the last item as well.

LINKED LIST OPERATIONS

1. **Traverse**
2. **Insert**
3. **Delete**

How to Traverse a Linked List

- Displaying the contents of a linked list is very simple.
- We keep moving the temp node to the next one and display its contents.
- When temp is *NULL*, we know that we have reached the end of the linked list so we get out of the while loop.



How to Add Elements to a Linked List

- You can add elements to the beginning, middle or end of the linked list.

Add to the beginning

- Allocate memory for new node
- Store data
- Change next of new node to point to head
- Change head to point to recently created node

Add to the End

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

Add to the Middle

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

How to Delete from a Linked List

- You can delete either from the beginning, end or from a particular position.

Delete from beginning

- Point head to the second node

Delete from end

- Traverse to second last element
- Change its next pointer to null

Delete from middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain