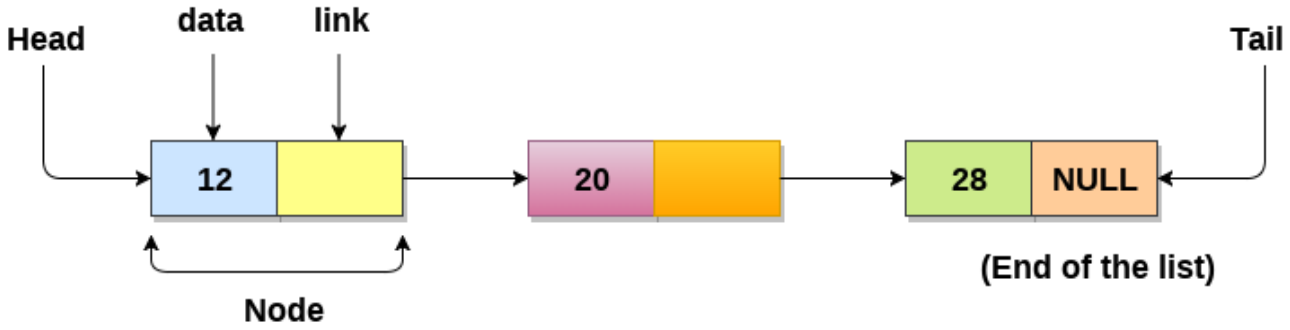
DATA STRUCTURES
UNIT-II

Syllabus: Linked Lists: Singly linked lists, representation and operations, doubly linked lists and circular linked lists, Comparing arrays and linked lists, Applications of linked lists.

Introduction to Linked List

- **What is Linked List?**
- Linked list is a linear data structure.
- Linked List can be defined as collection of objects called **nodes** that are randomly (non-contiguous) stored in the memory.
- A node contains two fields i.e. **data** stored at that particular address and **the pointer** which contains the address of the next node in the memory.
- The last node of the list contains pointer to the NULL.

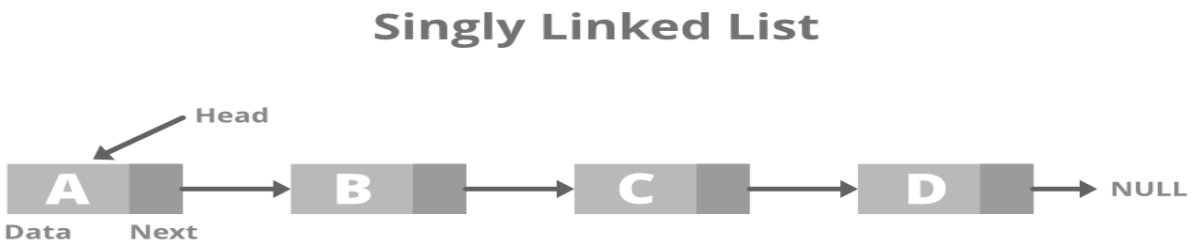


- **Advantages or uses or pros of a linked list:**
- **Optimized space utilization:** The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a list. This achieves **optimized utilization of space.**
- **Run-time memory allocation:** List size is limited to the memory size and doesn't need to be declared in advance.
- **Empty node cannot be present** in the linked list.
- We can **store values of primitive types or objects** in the singly linked list.
- **Disadvantages of Linked Lists**
- Sometimes the memory gets wasted because pointers need extra memory for storage.
- We can access elements in sequence. You cannot do this process in a random manner.
- In a linked list, reverse traversing is challenging.
- **Types of Linked List:** There are 3 different types of Linked Lists,

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

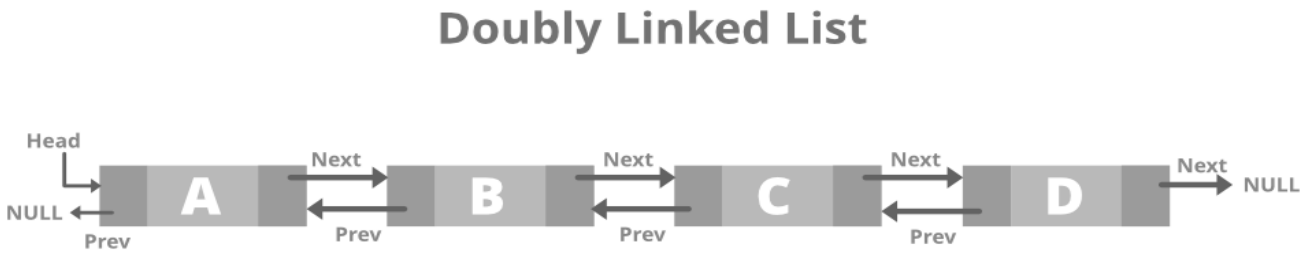
1. **Singly linked list:**

- It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.
- The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows the traversal of data only in one way. Below is the image for the same:



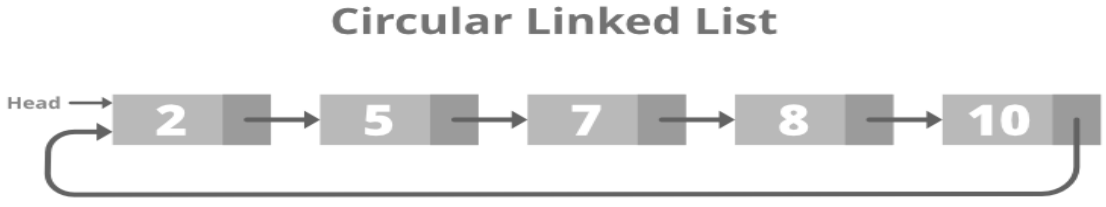
2. Doubly Linked List:

- A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence.
- Therefore, it contains three parts of data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well. Below is the image for the same:



3. Circular Linked List:

- A circular linked list is that in which the last node contains the pointer to the first node of the list.
- While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward and backward until we reach the same node we started. Thus, a circular linked list has no beginning and no end. Below is the image for the same:



Applications of Linked Lists

- **Applications of linked list in computer science:**
- **Implementation of stacks and queues**
- **Implementation of graphs**: Adjacency list representation of graphs is the most popular which uses a linked list to store adjacent vertices.
- **Dynamic memory allocation:** We use a linked list of free blocks.
- Maintaining a directory of names
- Performing arithmetic operations on long integers
- Manipulation of polynomials by storing constants in the node of the linked list.
- Representing sparse matrices.
- **Applications of linked list in the real world:**
- **Image viewer** – Previous and next images are linked and can be accessed by the next and previous buttons.
- **Previous and next page in a web browser** – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.
- **Music Player –** Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.
- **GPS navigation systems-** Linked lists can be used to store and manage a list of locations and routes, allowing users to easily navigate to their desired destination.
- **Robotics-** Linked lists can be used to implement control systems for robots, allowing them to navigate and interact with their environment.
- **Task Scheduling-** Operating systems use linked lists to manage task scheduling, where each process waiting to be executed is represented as a node in the list.
- **Image Processing-** Linked lists can be used to represent images, where each pixel is represented as a node in the list.
- **File Systems**- File systems use linked lists to represent the hierarchical structure of directories, where each directory or file is represented as a node in the list.
- **Symbol Table-** Compilers use linked lists to build a symbol table, which is a data structure that stores information about identifiers used in a program**.**
- **Undo/Redo Functionality-** Many software applications implement undo/redo functionality using linked lists, where each action that can be undone is represented as a node in a doubly linked list.
- **Speech Recognition-** Speech recognition software uses linked lists to represent the possible phonetic pronunciations of a word, where each possible pronunciation is represented as a node in the list.

- **Polynomial Representation-** Polynomials can be represented using linked lists, where each term in the polynomial is represented as a node in the list.
- **Simulation of Physical Systems-** Linked lists can be used to simulate physical systems, where each element in the list represents a discrete point in time and the state of the system at that time.
- **Applications of Circular Linked Lists:**
- Useful for implementation of a queue. Unlike this implementation, we don't need to maintain two-pointers for the front and rear if we use a circular linked list. We can maintain a pointer to the last inserted node and the front can always be obtained as next of last.
- Circular lists are useful in applications to go around the list repeatedly. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
- Circular Doubly Linked Lists are used for the implementation of advanced data structures like the Fibonacci Heap.
- Circular linked lists can be used to implement circular queues, which are often used in operating systems for scheduling processes and managing memory allocation.
- Used in database systems to implement linked data structures, such as B+ trees, which are used to optimize the storage and retrieval of data.
- Circular linked lists can be used in networking. For instance, to implement circular buffers for streaming data, such as video and audio, in networking applications.
- Video games use circular linked lists to manage sprite animations. Each frame of the animation is represented as a node in the list, and the last frame is connected to the first frame to create a loop.
- Circular linked lists can be used to represent a buffer of audio or signal data in signal processing applications. The last node is connected to the first node to create a loop, and the processing algorithms can efficiently iterate over the data.
- Traffic light control systems use circular linked lists to manage the traffic light cycles. Each phase of the traffic light cycle is represented as a node in the list, and the last node is connected to the first node to create a loop.

- **Application of Doubly Linked Lists:**
- Redo and undo functionality.
- Use of the Back and forward button in a browser.
- The most recently used section is represented by the Doubly Linked list.
- Other Data structures like Stack, Hash Table, and Binary Tree can also be applied by Doubly Linked List.
- Used to implement game objects and their interactions in a game engine.
- Used in networking.
- Used in Graph algorithms.
- Operating systems use doubly linked lists to manage the process scheduling. Each process waiting to be executed is represented as a node in the doubly linked list, and the operating system can easily traverse the list in both directions to manage the process queue.
- **Differentiate Array and Linked List:**

ARRAY	LINKED LIST
1. Arrays are stored in contiguous memory locations.	1. Linked lists are stored at non-contiguous memory locations.
2. Fixed in size.	2. Dynamic in size.
3. Memory is allocated at compile time.	3. Memory is allocated at run-time.
4. Uses less memory than linked lists.	4. Uses more memory as it stores both data and address of the next node.
5. Elements are accessed easily.	5. Element accessing requires the traversal of whole linked list.
6. Insertion and deletion operations take time.	6. Insertion and deletion operations are faster.
7. Arrays are implemented in C using loops.	7. Linked lists are implemented using self-referential structures and pointers.
8. Types of arrays: 1D, 2D, 3D.	8. Types of Linked lists: Singly, Doubly, Circular Linked lists.
9. Advantages of arrays: code optimization, random access, easy to operate, contiguous memory allocation etc.	9. Advantages of linked lists: Dynamic data structure, No memory wastage, Flexible, Efficient for large data, Scalability etc
10.Disadvantages of arrays: fixed size, inefficient memory utilization, slow in doing insertion and deletion operations etc.	10.Disadvantages of Linked lists: Memory usage, Complex implementation, Difficult to share data, Not suited for small datasets.
11. Applications of arrays: Used to store values of the same data type, Used in mathematical problems, To implement Search Algorithms, To implement Sorting Algorithms, Data structures. etc.	11. Applications of linked lists: Image viewer, GPS navigation systems, symbol table, music player, speech recognition, file systems, task scheduling , robotics, polynomial representation. Etc.

Advantages Of Linked List:

- **Dynamic data structure:** A linked list is a dynamic arrangement so it can grow and shrink at runtime by allocating and deallocating memory. So there is no need to give the initial size of the linked list.
- **No memory wastage**: In the Linked list, efficient memory utilization can be achieved since the size of the linked list increase or decrease at run time so there is no memory wastage and there is no need to pre-allocate the memory.
- **Implementation:** Linear data structures like stacks and queues are often easily implemented using a linked list.
- **Insertion and Deletion Operations:** Insertion and deletion operations are quite easier in the linked list. There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.
- **Flexible:** This is because the elements in Linked List are not stored in contiguous memory locations unlike the array.
- **Efficient for large data:** When working with large datasets linked lists play a crucial role as it can grow and shrink dynamically.
- **Scalability:** Contains the ability to add or remove elements at any position.

Disadvantages Of Linked List:

- **Memory usage**: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.
- **Traversal:** In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index. For example, for accessing a node at position n, one has to traverse all the nodes before it.
- **Reverse Traversing**: In a singly linked list reverse traversing is not possible, but in the case of a doubly-linked list, it can be possible as it contains a pointer to the previously connected nodes with each node. For performing this extra memory is required for the back pointer hence, there is a wastage of memory.
- **Random Access:** Random access is not possible in a linked list due to its dynamic memory allocation.
- **Lower efficiency at times:** For certain operations, such as searching for an element or iterating through the list, can be slower in a linked list.
- **Complex implementation**: The linked list implementation is more complex when compared to array. It requires a complex programming understanding.
- **Difficult to share data:** This is because it is not possible to directly access the memory address of an element in a linked list.
- **Not suited for small dataset**: Cannot provide any significant benefits on small dataset compare to that of an array.

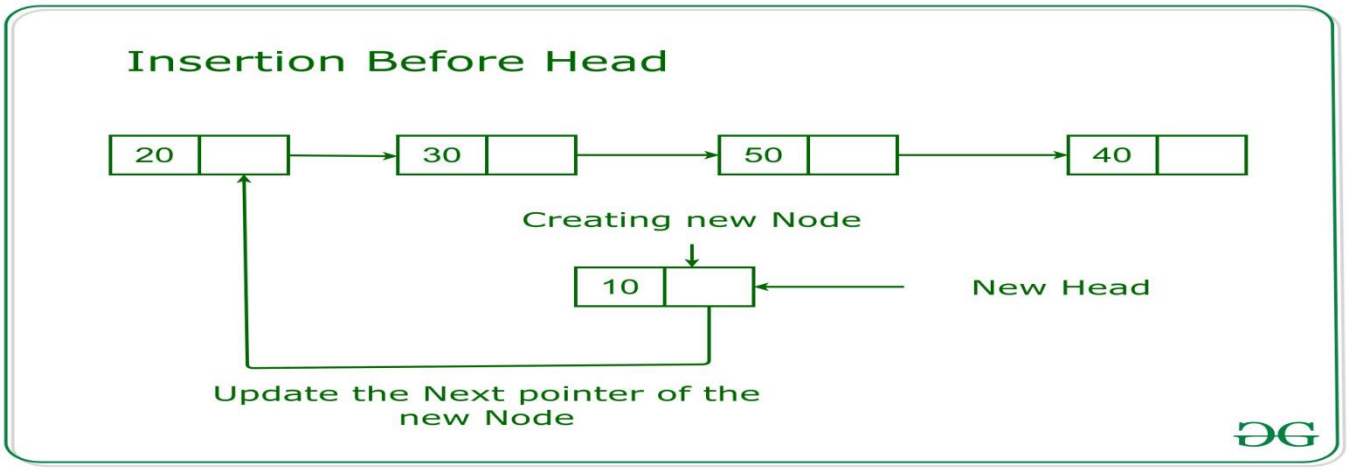
Operations on Singly Linked List: There are various operations which can be performed on singly linked list. A list of all such operations is given below.

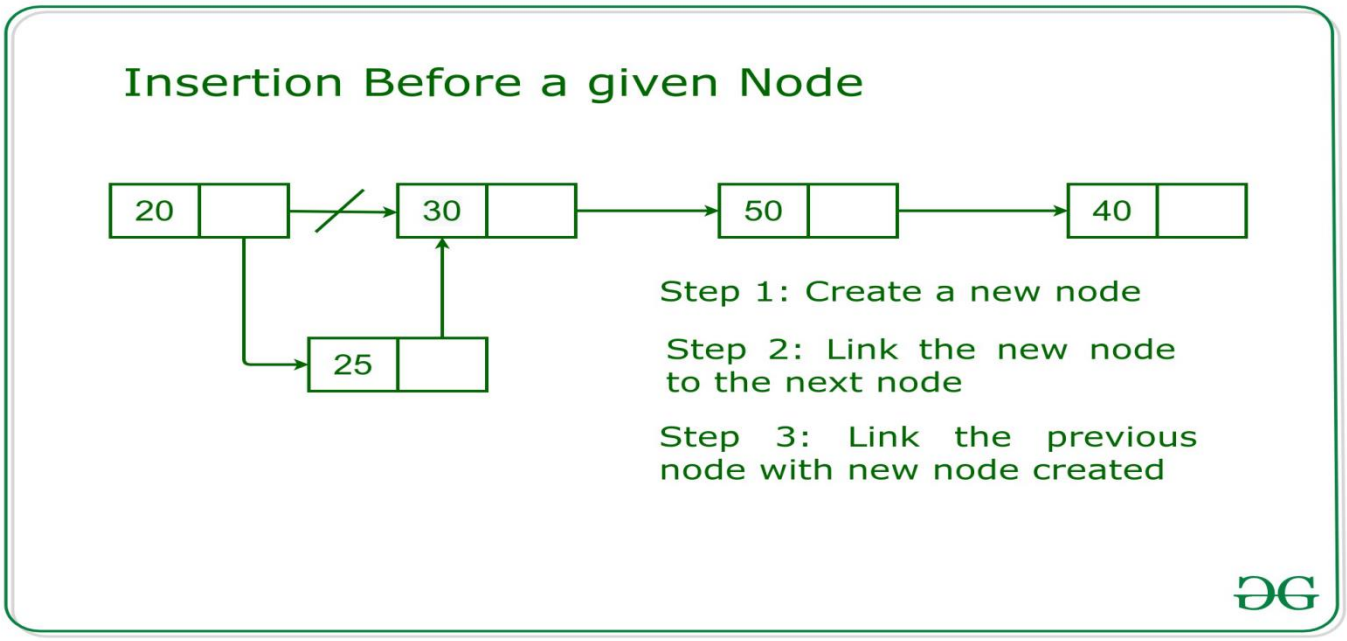
Node Creation

```
1.    struct node
2.    {
3.       int data;
4.       struct node *next;
5.    };
6.    struct node *head, *ptr;
7.    ptr = (struct node *)malloc(sizeof(struct node *));
```

Insertion: The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

SN	Operation	Description
1	Insertion at beginning	It involves inserting any element at the front of the list. We just need to a few link adjustments to make the new node as the head of the list.
2	Insertion at end of the list	It involves insertion at the last of the linked list. The new node can be inserted as the only node in the list or it can be inserted as the last one. Different logics are implemented in each scenario.
3	Insertion after specified node	It involves insertion after the specified node of the linked list. We need to skip the desired number of nodes in order to reach the node after which the new node will be inserted. .





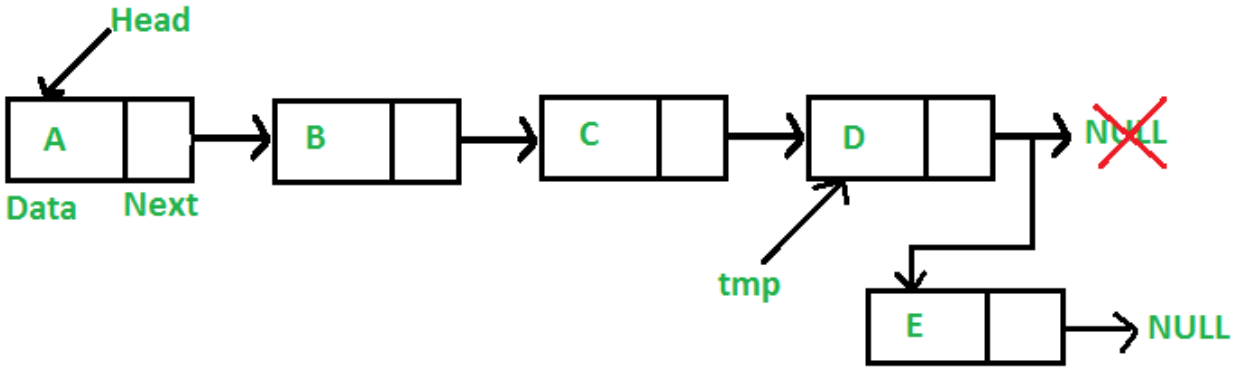
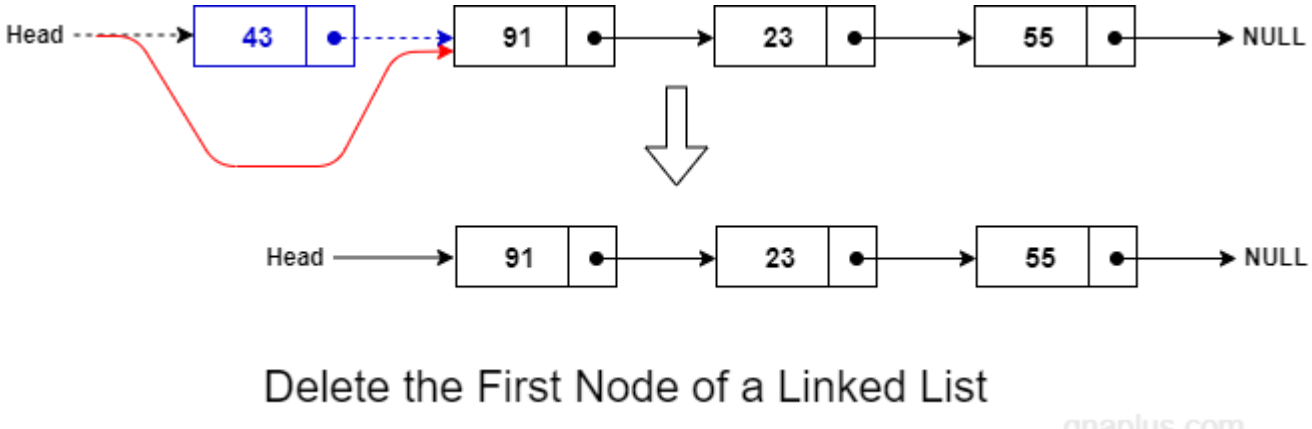
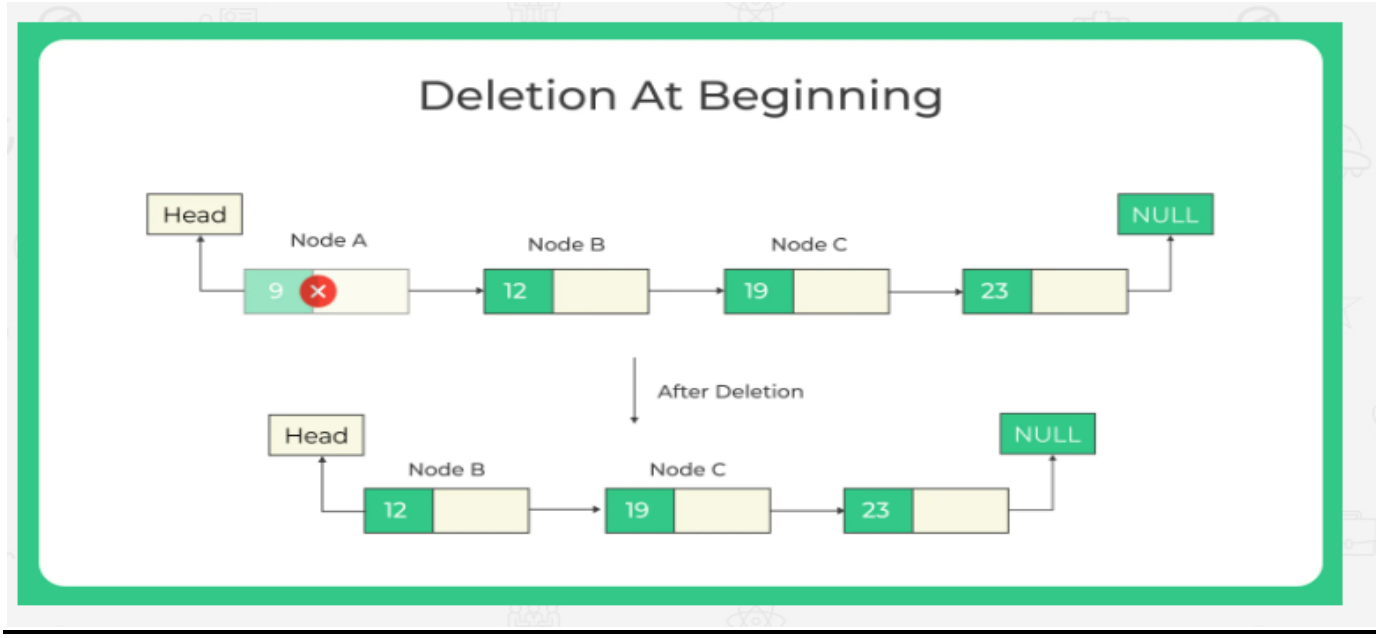
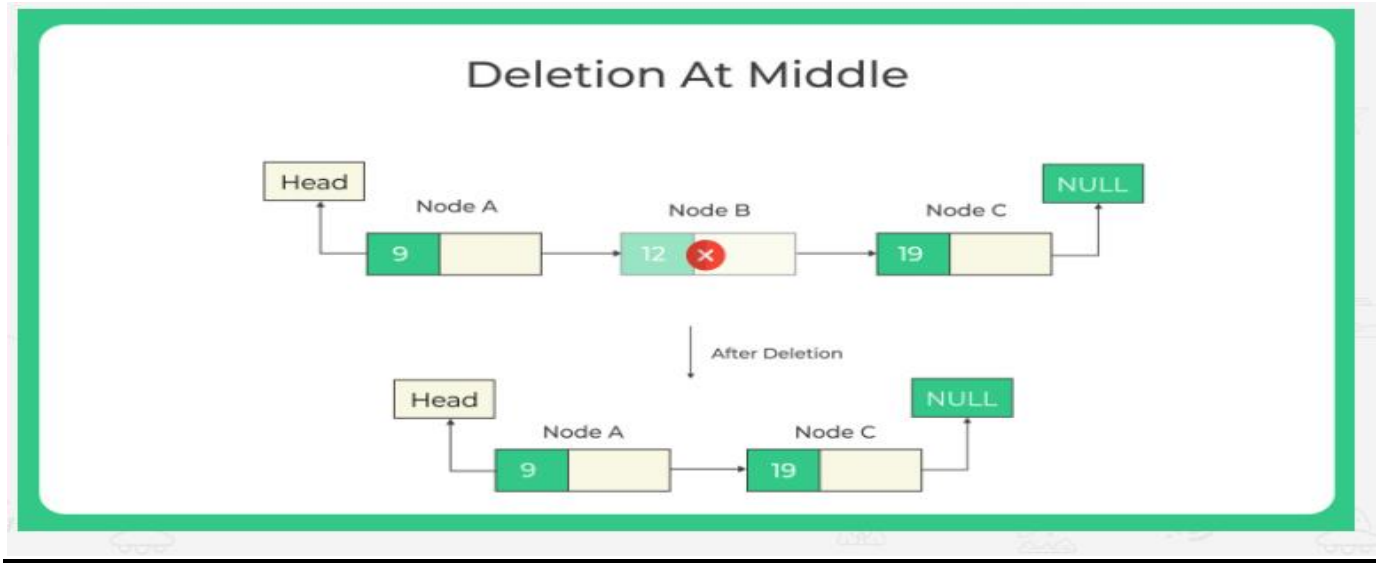


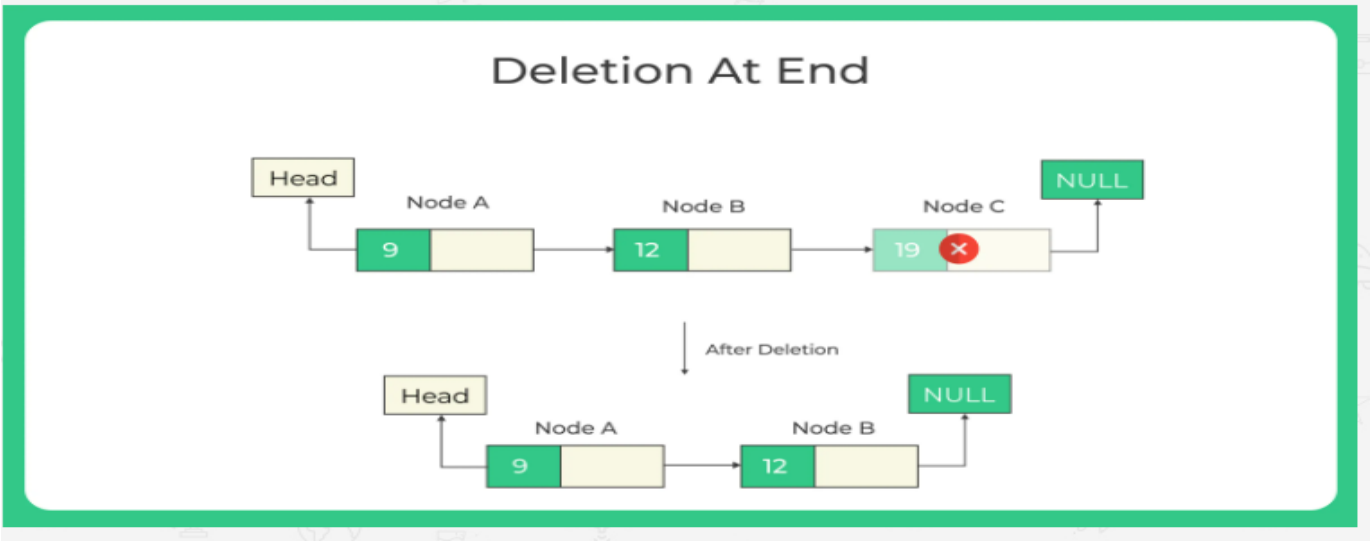
Fig. insertion of a new node at the end of the linked list

Deletion and Traversing: The Deletion of a node from a singly linked list can be performed at different positions. Based on the position of the node being deleted, the operation is categorized into the following categories.




Deletion At Beginning
Head
Node A
Node B
Node C
NULL
9
12
19
23
After Deletion
Head
Node B
Node C
NULL
12
19
23


Deletion At Middle
Head
Node A
Node B
Node C
NULL
9
12
19
After Deletion
Head
Node A
Node C
NULL
9
19


Deletion At End
Head
Node A
Node B
Node C
NULL
9
12
19
After Deletion
Head
Node A
Node B
NULL
9
12

SN	Operation	Description
1	Deletion at beginning	It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers.
2	Deletion at the end of the list	It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios.
3	Deletion after specified node	It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list.
4	Traversing	In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.
5	Searching	In searching, we match each element of the list with the given element. If the element is found on any of the location then location of that element is returned otherwise null is returned. .

Complexity

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Singly Linked List	θ(n)	θ(n)	θ(1)	θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

PARAMETER	SINGLY LINKED LIST	DOUBLY LINKED LIST	CIRCULAR LINKED LIST
Node Structure	Node contains 2 parts: data and link to next node.	Node contains 3 parts: data and links to previous and next nodes.	Node contains 2 parts: data and link to next node.
Traversing	Only forward traversing is allowed.	Both forward and backward traversing is allowed.	Only forward traversing is allowed but can jump last node to first.
Memory	It uses less memory per node (one pointer).	It uses more memory per node (two pointers).	It uses less memory per node (one pointer).
Use	Mostly used for stacks.	Used for stacks, heaps and binary trees.	Used to round robin scheduling.
Complexity	O(n)->insertion O(n)->deletion	O(n)->insertion O(n)->deletion	O(n)->insertion O(1)->deletion

Implementation	struct Node
	{
	int Data;
	struct Node *Next;
	};
		struct Node
		{
		int Data;
		struct Node *Next;
		struct Node *Previous;
		};
			Singly circular:
			struct Node
			{
			int Data;
			struct Node *Next;
			};
			Doubly circular:
			struct Node
			{
			int Data;
			struct Node *Next;
			struct Node *Previous;
			};