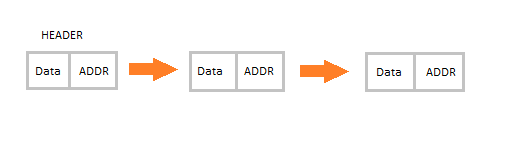
Introduction to Linked Lists

Linked List is a very commonly used linear data structure which consists of group of **nodes** in a sequence.

Each node holds its own **data** and the **address of the next node** hence forming a chain like structure.

Linked Lists are used to create trees and graphs.



Advantages of Linked Lists

* They are a dynamic in nature which allocates the memory when required.
* Insertion and deletion operations can be easily implemented.
* Stacks and queues can be easily executed.
* Linked List reduces the access time.

Disadvantages of Linked Lists

* The memory is wasted as pointers require extra memory for storage.
* No element can be accessed randomly; it has to access each node sequentially.
* Reverse Traversing is difficult in linked list.

Applications of Linked Lists

* Linked lists are used to implement stacks, queues, graphs, etc.
* Linked lists let you insert elements at the beginning and end of the list.
* In Linked Lists we don't need to know the size in advance.

Types of Linked Lists

There are 3 different implementations of Linked List available, they are:

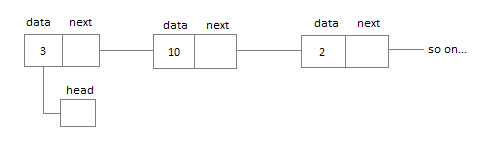
1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Let's know more about them and how they are different from each other.

Singly Linked List

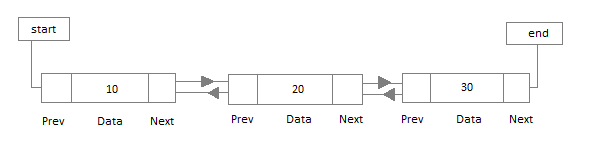
Singly linked lists contain nodes which have a **data** part as well as an **address part** i.e. next, which points to the next node in the sequence of nodes.

The operations we can perform on singly linked lists are **insertion**, **deletion** and **traversal**.



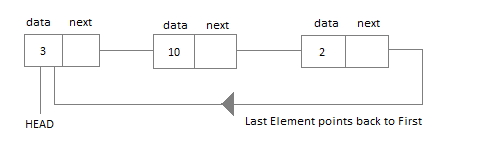
Doubly Linked List

In a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.



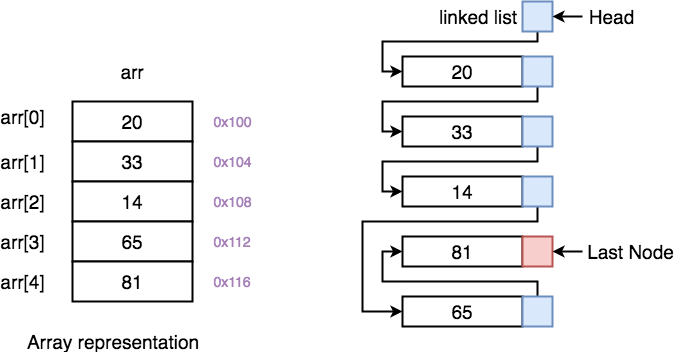
Circular Linked List

In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.



|  |  |
| --- | --- |
| **ARRAY** | **LINKED LIST** |
| Array is a collection of elements of similar data type. | Linked List is an ordered collection of elements of same type, which are connected to each other using pointers. |
| Array supports **Random Access**, which means elements can be accessed directly using their index, like arr[0] for 1st element, arr[6] for 7th element etc.  Hence, accessing elements in an array is **fast** with a constant time complexity of O(1). | Linked List supports **Sequential Access**, which means to access any element/node in a linked list, we have to sequentially traverse the complete linked list, upto that element.  To access **nth** element of a linked list, time complexity is O(n). |
| In an array, elements are stored in **contiguous memory location** or consecutive manner in the memory. | In a linked list, new elements can be stored anywhere in the memory.  Address of the memory location allocated to the new element is stored in the previous node of linked list, hence formaing a link between the two nodes/elements. |
| In array, **Insertion and Deletion** operation takes more time, as the memory locations are consecutive and fixed. | In case of linked list, a new element is stored at the first free and available memory location, with only a single overhead step of storing the address of memory location in the previous node of linked list.  Insertion and Deletion operations are **fast** in linked list. |
| Memory is allocated as soon as the array is declared, at **compile time**. It's also known as **Static Memory Allocation**. | Memory is allocated at **runtime**, as and when a new node is added. It's also known as **Dynamic Memory Allocation**. |
| In array, each element is independent and can be accessed using it's index value. | In case of a linked list, each node/element points to the next, previous, or maybe both nodes. |
| Array can **single dimensional**, **two dimensional** or **multidimensional** | Linked list can be **Linear(Singly)**, **Doubly** or **Circular** linked list. |
| Size of the array must be specified at time of array decalaration. | Size of a Linked list is variable. It grows at runtime, as more nodes are added to it. |
| Array gets memory allocated in the **Stack** section. | Whereas, linked list gets memory allocated in **Heap** section. |

Below we have a pictorial representation showing how consecutive memory locations are allocated for array, while in case of linked list random memory locations are assigned to nodes, but each node is connected to its next node using **pointer**.



### Why we need pointers in Linked List? [Deep Dive]

In case of array, memory is allocated in contiguous manner, hence array elements get stored in consecutive memory locations. So when you have to access any array element, all we have to do is use the array index, for example arr[4] will directly access the 5th memory location, returning the data stored there.

But in case of linked list, data elements are allocated memory at runtime, hence the memory location can be anywhere. Therefore to be able to access every node of the linked list, address of every node is stored in the previous node, hence forming a link between every node.

We need this additional **pointer** because without it, the data stored at random memory locations will be lost. We need to store somewhere all the memory locations where elements are getting stored.

Yes, this requires an additional memory space with each node, which means an additional space of O(n) for every n node linked list.

Linear Linked List

Linear Linked list is the default linked list and a linear data structure in which data is not stored in contiguous memory locations but each data node is connected to the next data node via a pointer, hence forming a chain.

The element in such a linked list can be inserted in 2 ways:

* Insertion at beginning of the list.
* Insertion at the end of the list.

Hence while writing the code for Linked List we will include methods to insert or add new data elements to the linked list, both, at the beginning of the list and at the end of the list.

We will also be adding some other useful methods like:

* Checking whether Linked List is empty or not.
* Searching any data element in the Linked List
* Deleting a particular Node(data element) from the List

Before learning how we insert data and create a linked list, we must understand the components forming a linked list, and the main component is the **Node**.

