<https://www.geeksforgeeks.org/what-is-linked-list/>

# What is Linked List

* **Difficulty Level :** [Easy](https://www.geeksforgeeks.org/easy/)
* **Last Updated :** 14 Sep, 2022

 Read

 Discuss

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. They include a series of connected nodes. Here, each node stores the data and the address of the next node.



Linked-List

## ****Why Linked List?****

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

* **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 upper limit irrespective of the usage.
* **Insertion of a new element / Deletion of a existing element in an array of elements is expensive:** The room has to be created for the new elements and to create room existing elements have to be shifted but in Linked list if we have the head node then we can traverse to any node through it and insert new node at the required position.

**Example:**   
In a system, if we maintain a sorted list of IDs in an array id[] = [1000, 1010, 1050, 2000, 2040].   
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 until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved due to this so much work is being done which affects the efficiency of the code.

[](https://practice.geeksforgeeks.org/courses/dsa-self-paced)

## ****Advantages of Linked Lists over arrays:****

* Dynamic Array.
* Ease of Insertion/Deletion.

## ****Drawbacks of Linked Lists:****

* Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a [binary search with linked lists](https://www.geeksforgeeks.org/binary-search-on-singly-linked-list/) efficiently with its default implementation.
* Extra memory space for a pointer is required with each element of the list.
* Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

## Types of Linked Lists:

* **Simple Linked List** – In this type of linked list, one can move or traverse the linked list in only one direction
* **Doubly Linked List** – In this type of linked list, one can move or traverse the linked list in both directions (Forward and Backward)
* **Circular Linked List** – In this type of linked list, the last node of the linked list contains the link of the first/head node of the linked list in its next pointer and the first/head node contains the link of the last node of the linked list in its prev pointer

## Basic operations on Linked Lists:

* [Deletion](https://www.geeksforgeeks.org/linked-list-set-3-deleting-node/)
* [Insertion](https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)
* [Search](https://www.geeksforgeeks.org/search-an-element-in-a-linked-list-iterative-and-recursive/)
* Display

## ****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 of the linked list. If the linked list is empty, then the value of the head points to NULL.

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

* A Data Item (we can store integer, strings, or any type of data).
* Pointer (Or Reference) to the next node (connects one node to another) or An address of another node

In C, we can represent a node using structures. Below is an example of a linked list node with integer data.   
In Java or C#, LinkedList can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.

|  |
| --- |
| class LinkedList {      Node head; // head of the 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;              next = null;          }      }  } |

#### Construction of a simple linked list with 3 nodes:

|  |
| --- |
| // A simple Java program to introduce 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      }        /\* 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 references to these three blocks as head,            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 the first Node refers to the second.          So they both are linked.             llist.head        second              third              |                |                  |              |                |                  |          +----+------+     +----+------+     +----+------+          | 1  |  o-------->| 2  | null |     |  3 | null |          +----+------+     +----+------+     +----+------+ \*/            second.next              = third; // Link second node with the third node            /\*  Now next of the second Node refers to third.  So          all three nodes are linked.             llist.head        second              third              |                |                  |              |                |                  |          +----+------+     +----+------+     +----+------+          | 1  |  o-------->| 2  |  o-------->|  3 | null |          +----+------+     +----+------+     +----+------+ \*/      }  } |

## ****Traversal of a Linked List****

In the previous program, we 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.

## [We strongly recommend that you click here and practice it, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/print-linked-list-elements/1)

|  |
| --- |
| // 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)          {              this.data = d;              next = null;          } // Constructor      }        /\* This function prints contents of linked list starting       \* from head \*/      public void printList()      {          Node n = head;          while (n != null) {              System.out.print(n.data + " ");              n = n.next;          }      }        // Driver's code      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 second node with the third node            // Function call          llist.printList();      }  } |

**Output**

1 2 3

**Time Complexity:**

|  |  |  |
| --- | --- | --- |
| **Time Complexity** | **Worst Case** | **Average Case** |
| **Search** | O(n) | O(n) |
| **Insert** | O(1) | O(1) |
| **Deletion** | O(1) | O(1) |

**Auxiliary Space:** O(N)