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 as shown in the below image:*



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.

**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.*

**Advantages of Linked Lists over arrays:**

* Dynamic Array.
* Ease of Insertion/Deletion.
* Insertion at the beginning is a constant time operation and takes O(1) time, as compared to arrays where inserting an element at the beginning takes O(n) time,where n is the number of elements in the array.

**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 the locality of reference which is not there in the case of linked lists.
* It takes a lot of time in traversing and changing the pointers.
* Reverse traversing is not possible in singly linked lists.
* It will be confusing when we work with pointers.
* Direct access to an element is not possible in a linked list as in an array by index.
* Searching for an element is costly and requires O(n) time complexity.
* Sorting of linked lists is very complex and costly.
* Appending an element to a linked list is a costly operation, and takes O(n) time, where n is the number of elements in the linked list, as compared to arrays that take O(1) time.

**Types of Linked Lists:**

* [**Simple Linked List**](https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/) – In this type of linked list, one can move or traverse the linked list in only one direction. where the next pointer of each node points to other nodes but the next pointer of the last node points to NULL. It is also called *“***Singly Linked List”**.
* [**Doubly Linked List**](https://www.geeksforgeeks.org/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**](https://www.geeksforgeeks.org/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.
* [**Doubly Circular Linked List**](https://www.geeksforgeeks.org/doubly-circular-linked-list-set-1-introduction-and-insertion/) – A Doubly Circular linked list or a circular 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 the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node.
* [**Header Linked List**](https://www.geeksforgeeks.org/header-linked-list-in-c/) – A header linked list is a special type of linked list that contains a header node at the beginning of the list.

**Representation of Singly 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 integers, strings, or any type of data).
* Pointer (Or Reference) to the next node (connects one node to another) or An address of another node.
* **class** Node {
* **public**:
* **int** data;
* Node\* next;
* };

## ****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.

// A simple C++ program for

// traversal of a linked list

#include <bits/stdc++.h>

**using** **namespace** std;

**class** Node {

**public**:

**int** data;

    Node\* next;

};

// This function prints contents of linked list

// starting from the given node

**void** printList(Node\* n)

{

**while** (n != NULL) {

        cout << n->data << " ";

        n = n->next;

    }

}

// Driver's code

**int** main()

{

    Node\* head = NULL;

    Node\* second = NULL;

    Node\* third = NULL;

    // allocate 3 nodes in the heap

    head = **new** Node();

    second = **new** Node();

    third = **new** Node();

    head->data = 1; // assign data in first node

    head->next = second; // Link first node with second

    second->data = 2; // assign data to second node

    second->next = third;

    third->data = 3; // assign data to third node

    third->next = NULL;

    // Function call

    printList(head);

**return** 0;

}

**Time Complexity:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Time Complexity** | **Worst Case** | **Average Case** |  |  |  |  |  |  |  |  | **Space complexity** |
| **Search** | O(n) | O(n) |  |  |  |  |  |  |  |  | O(1) |
| **Insertion** | O(1) | O(1) |  |  |  |  |  |  |  |  | O(1) |
| **Deletion** | O(1) | O(1) |  |  |  |  |  |  |  |  | O(1) |

* *Search* is O(n) because as data is not stored in contiguous memory locations so we have to traverse one by one.
* Insertion and Deletion are O(1) because we have to just link new nodes for Insertion with the previous and next node and dislink exist nodes for deletion from the previous and next nodes without any traversal.
* **Auxiliary Space:**O(N) [To store dynamic memory]

## Applications of Linked Lists

* ● Linked Lists can be used to implement useful data structures like stacks and queues.
* ● Linked Lists can be used to implement hash tables, each bucket of the hash table can be a linked list.
* ● Linked Lists can be used to implement graphs (Adjacency List representation of graph).
* ● Linked Lists can be used in a refined way in implementing different file systems in one form or another.