**Linked list Data Structure**

A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the **data** and the **address** of the next node. For example.



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.

Linked lists can be of multiple types: **singly**, **doubly**, and **circular linked list**. In this article

## Representation of Linked List

Let's see how each node of the linked list is represented. Each node consists:

* A data item
* An address of another node

## Linked List Utility

Lists are one of the most popular and efficient data structures, with implementation in every programming language like C, C++, Python, Java, and C#.

Apart from that, linked lists are a great way to learn how pointers work. By practicing how to manipulate linked lists, you can prepare yourself to learn more advanced data structures like graphs and trees.

## Linked List Applications

* Dynamic memory allocation
* Implemented in stack and queue
* In **undo** functionality of softwares
* Hash tables, Graphs

**Linked List Operations: Traverse, Insert and Delete**

In this tutorial, you will learn different operations on a linked list. Also, you will find implementation of linked list operations in C/C++, Python and Java.

There are various linked list operations that allow us to perform different actions on linked lists. For example, the insertion operation adds a new element to the linked list.

Here's a list of basic linked list operations that we will cover in this article.

* [Traversal](https://www.programiz.com/dsa/linked-list-operations#traverse) - access each element of the linked list
* [Insertion](https://www.programiz.com/dsa/linked-list-operations#add) - adds a new element to the linked list
* [Deletion](https://www.programiz.com/dsa/linked-list-operations#delete) - removes the existing elements
* [Search](https://www.programiz.com/dsa/linked-list-operations#search) - find a node in the linked list
* [Sort](https://www.programiz.com/dsa/linked-list-operations#sort) - sort the nodes of the linked list

### Things to Remember about Linked List

* head points to the first node of the linked list
* next pointer of the last node is NULL, so if the next current node is NULL, we have reached the end of the linked list.

## Insert Elements to a Linked List

### 1. Insert at 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

### 2. Insert at the End

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

### 3. Insert at 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

## Delete from a Linked List

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

### 1. Delete from beginning

* Point head to the second node

### 2. Delete from end

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

### 3. Delete from middle

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

## Search an Element on a Linked List

You can search an element on a linked list using a loop using the following steps. We are finding item on a linked list.

* Make head as the current node.
* Run a loop until the current node is NULL because the last element points to NULL.
* In each iteration, check if the key of the node is equal to item. If it the key matches the item, return true otherwise return false.

## Sort Elements of a Linked List

We will use a simple sorting algorithm, [Bubble Sort](https://www.programiz.com/dsa/bubble-sort), to sort the elements of a linked list in ascending order below.

1. Make the head as the current node and create another node index for later use.
2. If head is null, return.
3. Else, run a loop till the last node (i.e. NULL).
4. In each iteration, follow the following step 5-6.
5. Store the next node of current in index.
6. Check if the data of the current node is greater than the next node. If it is greater, swap current and index.