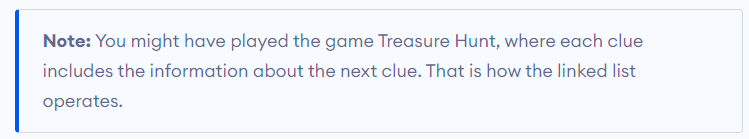
Linked list Data Structure

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,

|  |
| --- |
| linked list data structure |
| Linked list Data Structure |

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, we will focus on the **singly linked list**. To learn about other types, visit [Types of Linked List](https://www.programiz.com/dsa/linked-list-types).



Representation of Linked List

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

* A data items
* An address of another node

We wrap both the data item and the next node reference in a struct as:

struct node

{

int data;

struct node \*next;

};

Understanding the structure of a linked list node is the key to having a grasp on it.

Each struct node has a data item and a pointer to another struct node. Let us create a simple Linked List with three items to understand how this works.

|  |
| --- |
| // Linked list implementation in Java  class LinkedList {  // Creating a node  Node head;  static class Node {  int value;  Node next;  Node(int d) {  value = d;  next = null;  }  }  public static void main(String[] args) {  LinkedList linkedList = new LinkedList();  // Assign value values  linkedList.head = new Node(1);  Node second = new Node(2);  Node third = new Node(3);  // Connect nodess  linkedList.head.next = second;  second.next = third;  // printing node-value  while (linkedList.head != null) {  System.out.print(linkedList.head.value + " ");  linkedList.head = linkedList.head.next;  }  }  } |

In just a few steps, we have created a simple linked list with three nodes.

|  |
| --- |
| representing linked list by connecting each node with next node using address of next node |
| Linked list Representation |

The power of a linked list comes from the ability to break the chain and re-join­­­ it. E.g. if you wanted to put an element 4 between 1 and 2, the steps would be:­­­

* Create a new struct node and allocate memory to it.
* Add its data value as 4
* Point its next pointer to the struct node containing 2 as the data value
* Change the next pointer of "1" to the node we just created.

Doing something similar in an array would have required shifting the positions of all the subsequent elements.

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 Complexity

Time Complexity

|  |  |  |
| --- | --- | --- |
|  | Worst case | Average Case |
| Search | O(n) | O(n) |
| Insert | O(1) | O(1) |
| Deletion | O(1) | O(1) |

Space Complexity: O(n)

Linked List Applications

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