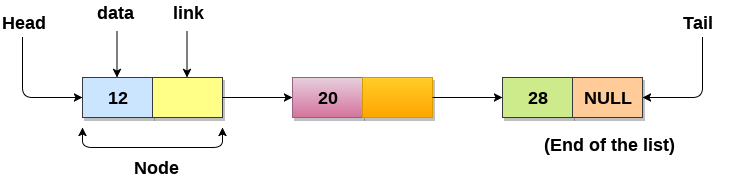
Linked List

* Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
* A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
* The last node of the list contains pointer to the null.



## **Uses of Linked List**

* The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
* list size is limited to the memory size and doesn't need to be declared in advance.
* Empty node can not be present in the linked list.
* We can store values of primitive types or objects in the singly linked list.

## **Why use linked list over array?**

Till now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known in order to decide the data structure which will be used throughout the program.

Array contains following limitations:

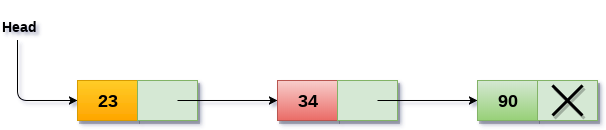
1. The size of array must be known in advance before using it in the program.
2. Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
3. All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

## **Singly linked list or One way chain**

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist of two parts: data part and link part. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

One way chain or singly linked list can be traversed only in one direction. In other words, we can say that each node contains only next pointer, therefore we can not traverse the list in the reverse direction.

Consider an example where the marks obtained by the student in three subjects are stored in a linked list as shown in the figure.



In the above figure, the arrow represents the links. The data part of every node contains the marks obtained by the student in the different subject. The last node in the list is identified by the null pointer which is present in the address part of the last node. We can have as many elements we require, in the data part of the list.

## **Operations on Singly Linked List**

There are various operations which can be performed on singly linked list. A list of all such operations is given below.

### **Node Creation**

1. struct node
2. {
3. **int** data;
4. struct node \*next;
5. };
6. struct node \*head, \*ptr;
7. ptr = (struct node \*)malloc(sizeof(struct node \*));

**Advantages over arrays**

**1)** Dynamic size  
**2)** Ease of insertion/deletion

**Drawbacks:**

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

# A simple Python program for traversal of a linked list

# Node class

class Node:

    # Function to initialise the node object

    def \_\_init\_\_(self, data):

        self.data = data  # Assign data

        self.next = None  # Initialize next as null

# Linked List class contains a Node object

class LinkedList:

    # Function to initialize head

    def \_\_init\_\_(self):

        self.head = None

    # This function prints contents of linked list

    # starting from head

    def printList(self):

        temp = self.head

        while (temp):

            print (temp.data)

            temp = temp.next

# Code execution starts here

if \_\_name\_\_=='\_\_main\_\_':

    # Start with the empty list

    llist = LinkedList()

    llist.head = Node(1)

    second = Node(2)

    third = Node(3)

    llist.head.next = second; # Link first node with second

    second.next = third; # Link second node with the third node

    llist.printList()

output : 1 2 3