**MY DSA DOCUMENT**

**1.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:  


**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.

**1.creating nodes**

**C**

// A linked list node

struct Node {

    int data;

    struct Node\* next;

};

**C++**

class Node {

public:

    int data;

    Node \*next;

};

**2. creating Linked list with 3 nodes**:

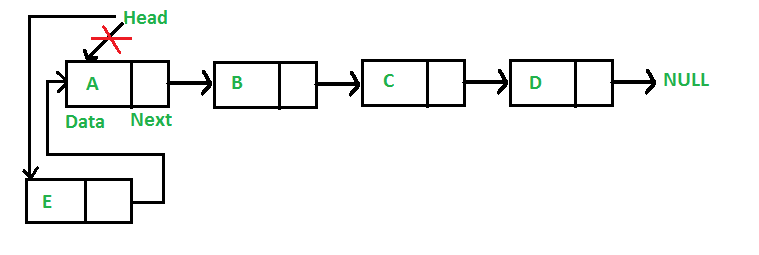
**Code**:<https://www.geeksforgeeks.org/linked-list-set-1-introduction/>

**3.Traversal**

**Code:**[**https://www.geeksforgeeks.org/linked-list-set-1-introduction/**](https://www.geeksforgeeks.org/linked-list-set-1-introduction/)

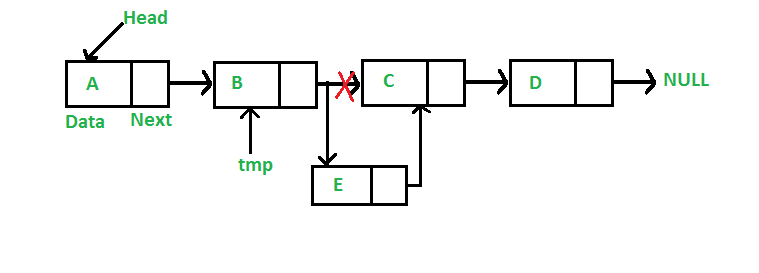
**4.Node insertion**

**(a) in front**



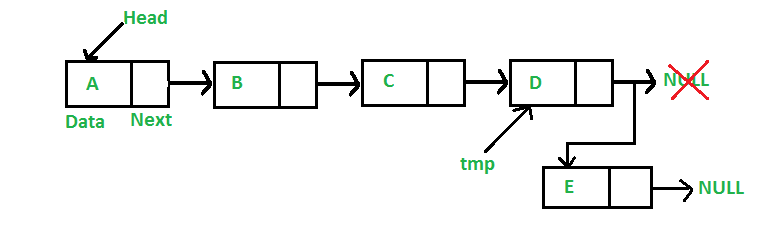
*code:*[*https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/*](https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)

**(b) add node after a given node**



*Code:*[*https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/*](https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)

**(c) at the end**



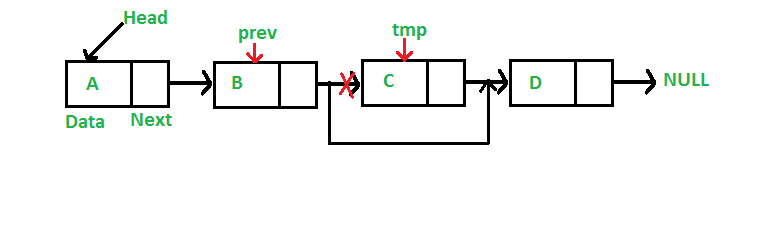
*Code:*[*https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/*](https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)

**d)Clubbed insertion codes**

Code:<https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/>

**5.Node deletion**

**a)Deletion of a particular node**



**Code:**[*https://www.geeksforgeeks.org/linked-list-set-3-deleting-node/*](https://www.geeksforgeeks.org/linked-list-set-3-deleting-node/)

**b)Delete a node from given position**

Code:<https://www.geeksforgeeks.org/delete-a-linked-list-node-at-a-given-position/>

**c)To delete whole list**

[*https://www.geeksforgeeks.org/write-a-function-to-delete-a-linked-list/*](https://www.geeksforgeeks.org/write-a-function-to-delete-a-linked-list/)

**2.CIRCULAR LINKED LIST**



**1.Traversal**

[**https://www.geeksforgeeks.org/circular-linked-list-set-2-traversal/**](https://www.geeksforgeeks.org/circular-linked-list-set-2-traversal/)

**2.all types of insertions**

**Code:**

<https://www.geeksforgeeks.org/circular-singly-linked-list-insertion/>

**3.DOUBLE LINKED LIST**

Advantages are-

We can traverse in both the directions

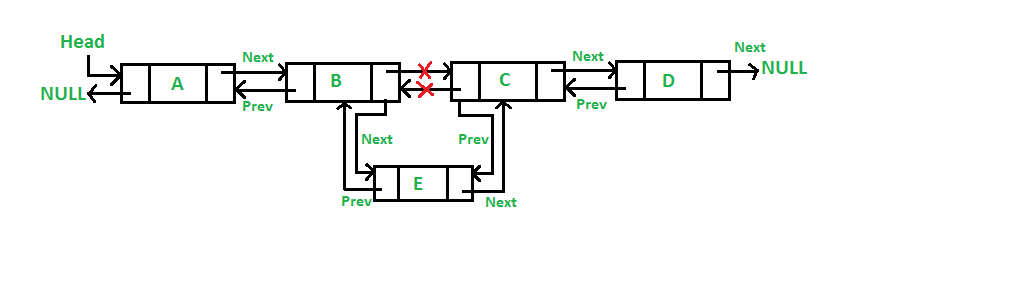
Easy to delete or insert node



**1)inserting node at front**

**Code:** [**https://www.geeksforgeeks.org/doubly-linked-list/**](https://www.geeksforgeeks.org/doubly-linked-list/)

**2)adding a node after a given node**



[**https://www.geeksforgeeks.org/doubly-linked-list/**](https://www.geeksforgeeks.org/doubly-linked-list/)

**3.Code for all approaches**

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

**Reverse a linked list**

<https://www.geeksforgeeks.org/reverse-a-linked-list/>