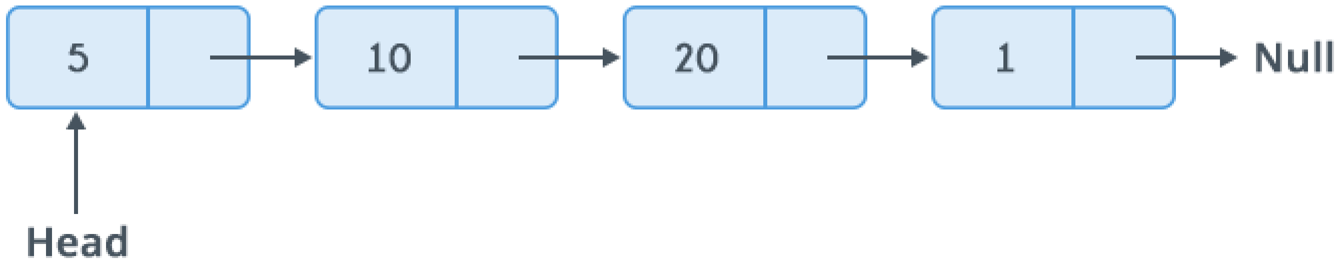
# LinkedList

## Singly linked List

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.



**Why Linked List?**
Arrays can be used to store linear data of similar types, but arrays have following limitations.

1. 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.
2. Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

**Advantages over arrays:**

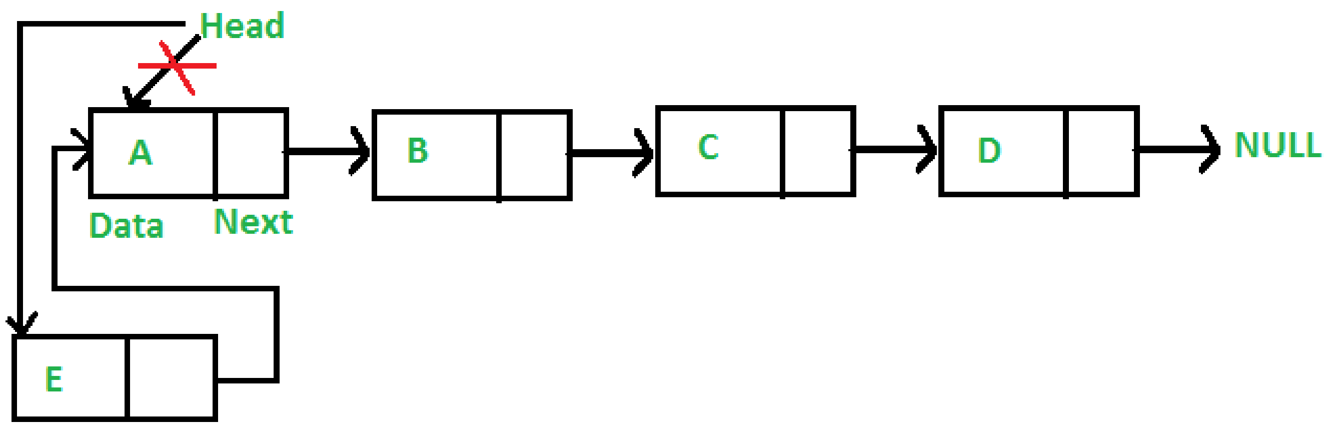
1. Dynamic size
2. Ease of insertion / deletion

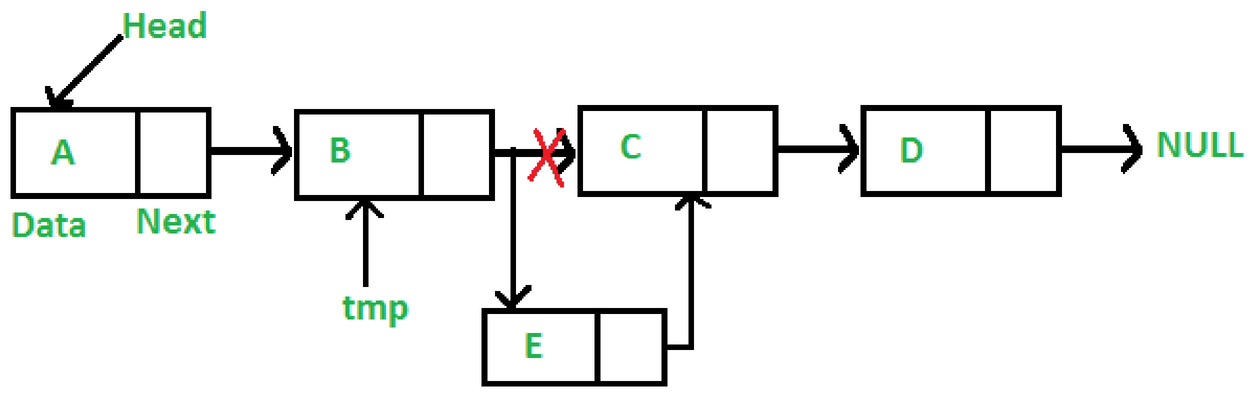
**Drawbacks:**

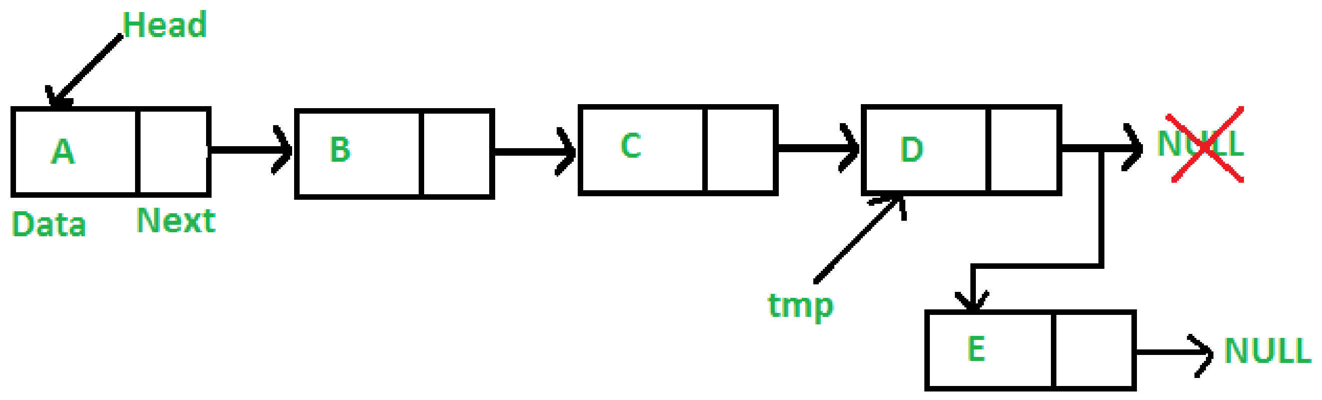
1. Random access is not allowed. We have to access elements sequentially starting from the first node.
   So we cannot do binary search with linked lists efficiently with its default implementation.
2. Extra memory space for a pointer is required with each element of the list.
3. Not cache friendly.
   **Explanation:** CPU caches actually do two things. First on is caching recently used memory. The other however is predicting which memory is going to be used in near future. The algorithm is usually quite simple - it assumes that the program processes big array of data and whenever it

accesses some memory it will pre-fetch few more bytes behind. This doesn't work for linked list as the nodes are randomly placed in memory. For linked list it reads one block and the rest may be wasted as the next node can be in completely different chunk of memory.

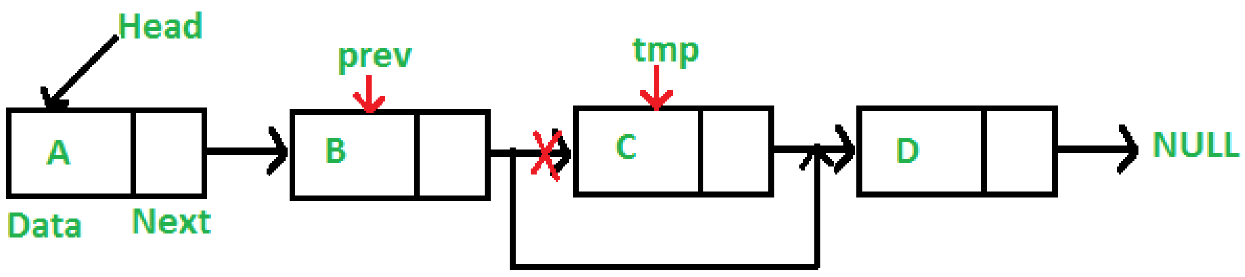
## Adding node to singly linked List







## Deleting node to singly linked List



## Doubly linked list

A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.

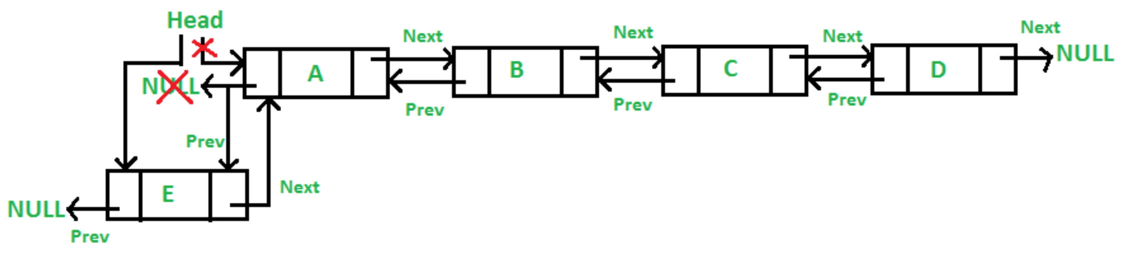
Advantages over singly linked list:

1. A DLL can be traversed in both forward and backward direction.
2. The delete operation in DLL is more efficient needing just pointer to the node to be deleted.
3. We can quickly insert a new node before a given node.
   **Explanation:** In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

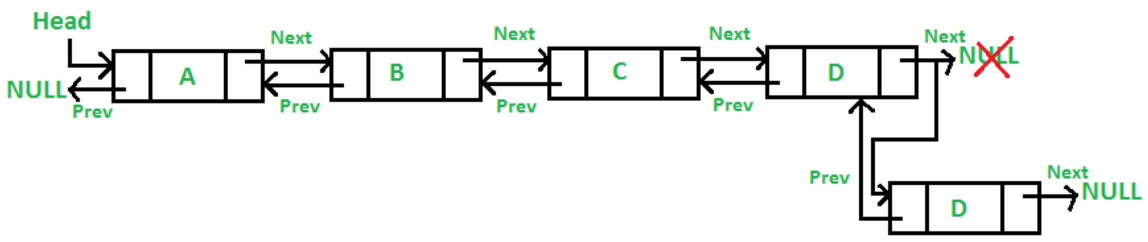
## Insertion in DLL

A node can be added in four ways:

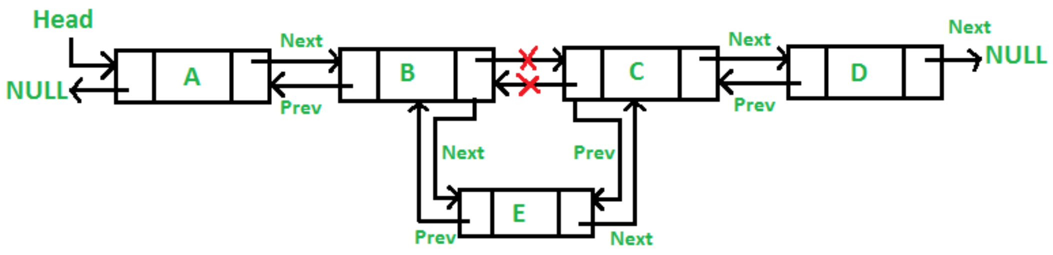
1. At the front of the DLL



2. At the end of the DLL



3. After a given node.



4. Before a given node.

## Deletion in DLL

Consider at least 3 cases:

1. Removing a node from the beginning.
2. Removing a node from the middle.
3. Removing a node from the end.

## Circular Linked List

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

Advantages of Circular Linked Lists:

1. Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
2. Useful for implementation of queue. We don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.

3. Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
4. Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.