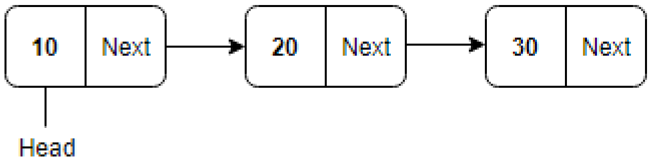
Lecture 3 -Linked List (1)

Linked List is a linear data structure which consists of a group of nodes in a sequence. Each node contains two parts.

- Data– Each node of a linked list can store a data.
- Address – Each node of a linked list contains an address to the next node, called "Next".

The first node of a Linked List is referenced by a pointer called Head

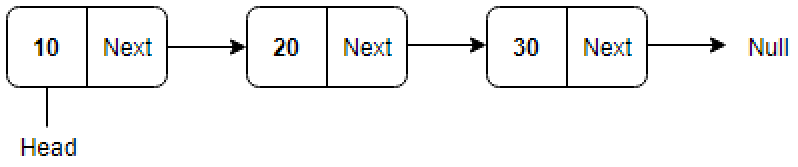


Advantages of Linked List

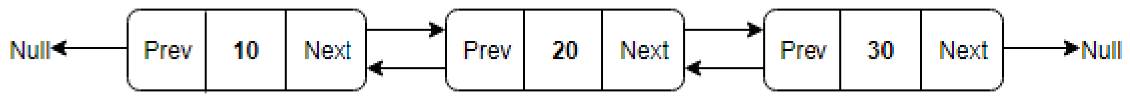
- They are dynamic in nature and allocate memory as and when required.
- Insertion and deletion is easy to implement.
- Other data structures such as Stack and Queue can also be implemented easily using Linked List.
- It has faster access time and can be expanded in constant time without memory overhead.
- Since there is no need to define an initial size for a linked list, hence memory utilization is effective.
- Backtracking is possible in doubly linked lists.

Types of Linked List

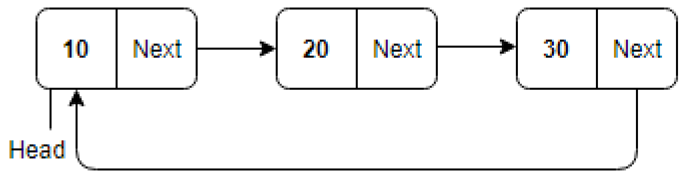
- **Singly Linked List:** Singly linked lists contain nodes which have a data part and an address part, i.e., Next, which points to the next node in the sequence of nodes. The next pointer of the last node will point to null.



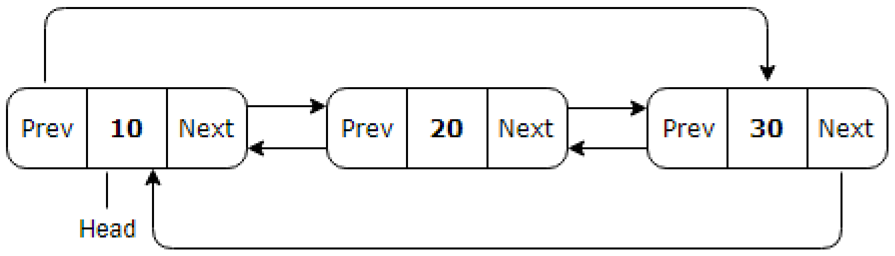
- **Doubly Linked List:** In a doubly linked list, each node contains two links - the first link points to the previous node and the next link points to the next node in the sequence.The prev pointer of the first node and next pointer of the last node will point to null.



- **Circular Linked List:** In the circular linked list, the next of the last node will point to the first node, thus forming a circular chain.



- **Doubly Circular Linked List:** In this type of linked list, the next of the last node will point to the first node and the previous pointer of the first node will point to the last node.



Creating a Linked List

The node of a singly linked list contains a data part and a link part. The link will contain the address of next node and is initialized to null. So, we will create class definition of node for singly linked list as follows -

```
class Node {
   int data;
   Node next;
  public Node(int d) {
        data = d;
        next = null;
    }
}
```

The node for a Doubly Linked list will contain one data part and two link parts - previous link and next link. Hence, we create a class definition of a node for the doubly linked list as shown below.

```
class DNode {
    int data;     DNode prev;      DNode next;
    public DNode(int d) {
        data = d;         prev = null;         next = null;
    }
}
```

Now, our node has been created, so, we will create a linked list class now. When a new Linked List is instantiated, it just has the head, which is Null.The SinglyLinkedList class will contain nodes of type Node class. Hence, SinglyLinkedList class definition will look like below.

```
class SingleLinkedList {
    Node head;
}
```

The DoublyLinkedList class will contain nodes of type DNode class. Hence, DoublyLinkedList class will look like this.

```
class DoubleLinkedList {
     DNode head;
}
```

Various operations on Linked list

Insert data at front of the Linked List

- The first node, head, will be null when the linked list is instantiated. When we want to add any node at the front, we want the head to point to it.
- We will create a new node. The next of the new node will point to the head of the Linked list.
- The previous Head node is now the second node of Linked List because the new node is added at the front. So, we will assign head to the new node.

```
void InsertFront(SingleLinkedList singlyList, int new_data) {

  Node new_node = new Node(new_data);

  new_node.next = singlyList.head;

  singlyList.head = new_node;

}
```

To insert the data at front of the doubly linked list, we have to follow one extra step .i.e point the previous pointer of head node to the new node. So, the method will look like this.

```
void InsertFront(DoubleLinkedList doubleLinkedList, int data) {

  DNode newNode = new DNode(data);

  newNode.next = doubleLinkedList.head;

  newNode.prev = null;

  if (doubleLinkedList.head != null) {

    doubleLinkedList.head.prev = newNode;

  }

  doubleLinkedList.head = newNode;
}
```

Insert data at the end of Linked List

- If the Linked List is empty, then we simply add the new node as the Head of the Linked List.
- If the Linked List is not empty, then we find the last node and make next of the last node to the new node, hence the new node is the last node now.

```
public void InsertLast(int new_data)
{
   Node new_node = new Node(new_data);
   if (head == null) {
      head = new_node;
      return;
   }
   Node lastNode = GetLastNode();
   lastNode.next = new_node;
}
```

To insert the data at the end of a doubly linked list, we have to follow one extra step; .i.e., point previous pointer of new node to the last node.so the method will look like this.

```
void InsertLast(DoubleLinkedList doubleLinkedList, int data) {
   DNode newNode = new DNode(data);
   if (doubleLinkedList.head == null) {
      newNode.prev = null;
      doubleLinkedList.head = newNode;
```

```
        return;
    }
    DNode lastNode = GetLastNode(doubleLinkedList);
    lastNode.next = newNode;
    newNode.prev = lastNode;
}
```

The last node will be the one with its next pointing to null. Hence we will traverse the list until we find the node with next as null and return that node as last node. Therefore the method to get the last node will be

```
Node GetLastNode() {
    Node temp =   head;
    while (temp.next != null) {
        temp = temp.next;
    }
    return temp;
}
```

In the above mentioned method, pass doubleLinkedList object to get last node for Doubly Linked List.