delete least used // if size exceeds.

Linked list :

Array = ghar ki line (fixed blocks)

linked list is like a train, connected

(data/ Next)

3 types of linked list:→ Sigular
→ doubly
→ reversed

Rules of LL:

1. head ko hamesha save
2. Null check first
3. One node case alag handle
4. pointer movement

ptr = ptr → next;

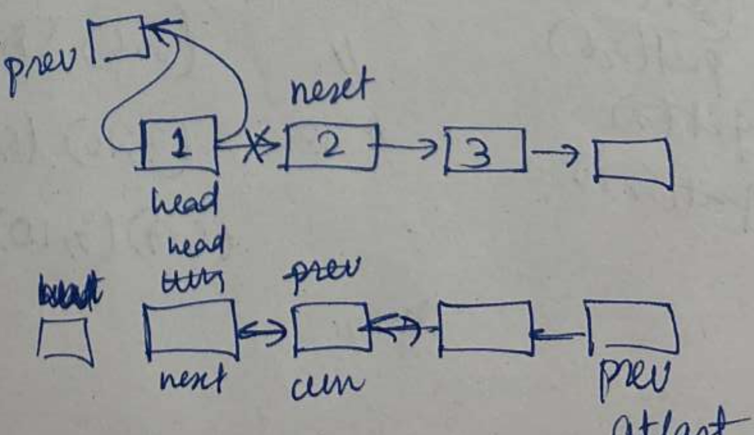
basic Node Structure:

```
struct Node {
      int data;
      Node* next;
   Node(int x) {
      data=x;
      next=Null;
   }
```

☆ Reverse a linked list:

TC → O(n)
SC → O(1)

```
Linkedlist* reverselist (ListNode* head)
{
   Linkedlist* prev=Null;
            * curr = head;
   while(curr != Null)
   {
      * Next= curr → next;
      curr → next = prev
      prev= curr;
      curr = next;
   }
   return prev;
}
```

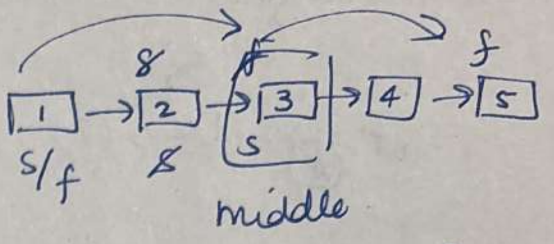


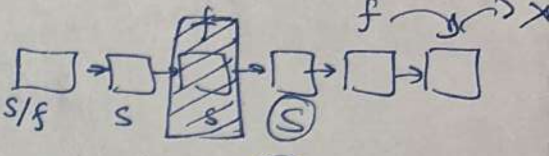
Middle of linked list :

```
listNode* middle( listNode* head)
{
  Node* slow = head;
  Node* fast = head;

  while(fast && fast->next)
  {
      slow = slow->next;
      fast = fast->next->next;
  }

  return slow;
}
```





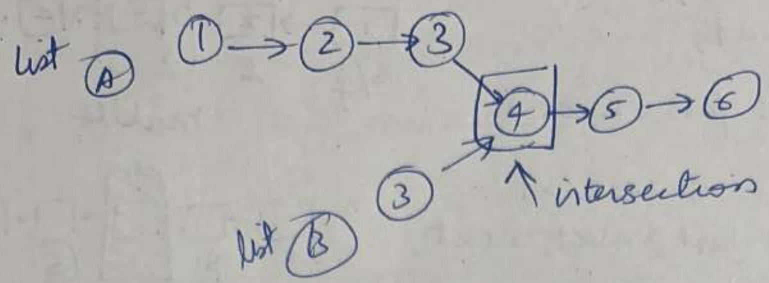
so, slow = 4

Detect cycle

* floyd's cycle detection

```
bool hascycle( listNode* head)
 {
   *slow = head;
   *fast = head;
   while(fast && fast->next)
   {
      slow = slow->next;
      fast = fast->next->next;
      if(slow == fast) return true;
   }
 }
 return false;
}
```

Intersection of linked list is the first node at which two singly linked list merge or start sharing the same nodes



A	B
1	3
2	4
3	5
4	6
5	Null
6	~~Null~~ 1
Null	2
3	3

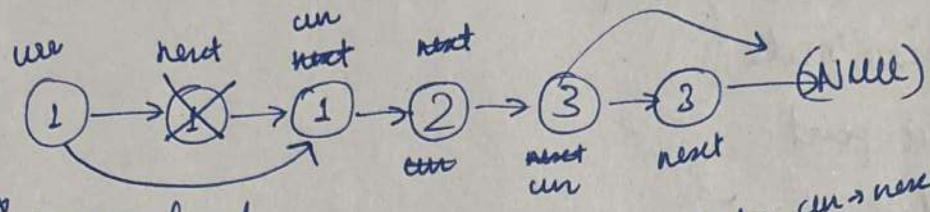
A	B
1	3
2	4
3	5
4	6
5	Null
6	1
Null	2
3	3
4	4

```
while( a != b)
{
    a = (a == Null) ? head B : a -> next
    b = (b == Null) ? head A : B -> next.
}
return A / B    both will be at intersection
```

agr, dono ko traverse karo, aaek bhi null hogya use dusere ka head assign kardo, then traverse,
aaak time they if they both equal will be at intersection

Delete Duplicate from a Sorted List

① Vector bana ke duplicate remove karke, LL vaps banado.



curr→next = curr→next→next;

```
ListNode* curr = head;
while(curr)
{
  while(curr→next)
  {
    if(curr.val = = curr→next→val)
    {
       curr→next = curr→next→next;
    }
  }
  curr = curr→next;
}
```

# go for (curr) while loop than go for (curr→next) while loop:
if (curr→val == curr→next→val) //duplicates.
Skip the node, connect curr to the next node.
i.e. (curr→next = curr→next→next)

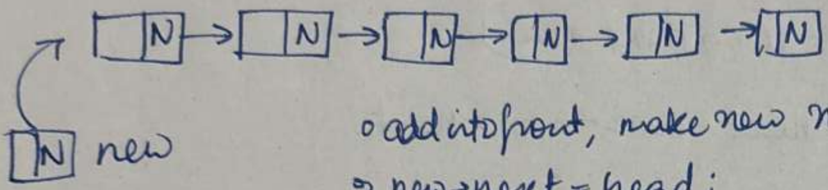
all set;
else { just iterate the node to curr = curr→next;

```
※ if(head== NULL) return NULL;
ListNode* curr=head;
while(curr && curr→next)
{
  if(curr→ val == curr→next →val)        // duplicate node
  {                                          skipped
    curr→next = curr→next →next;
  }
  else {
     curr= curr→next;                     // move only when
  }                                          no duplicate
}
```

Insertion in linked List

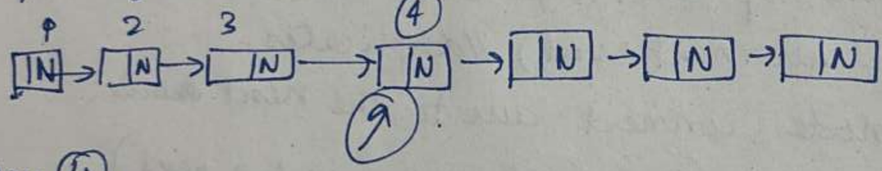
① at front
② before a given node
③ after a given node
④ at a specific point
⑤ At the end of LL; } Important.

Ⓐ At front.



o add into front, make new node,
⇒ new→next = head;
⇒ head = new;

Ⓑ before a given node



before ④.
→ traverse [Key = ④]
→ go to prev node;

```
Node* new = new node(val))
new→next = prev→next;
```

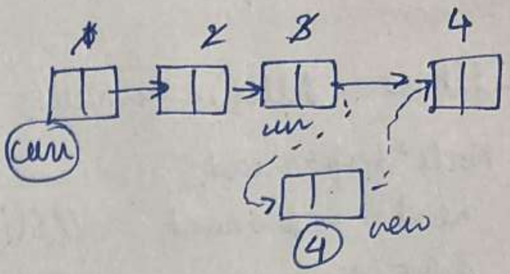
```
if (head == NULL) return head;
//if at head;
if(head→val == Key)
    insertathead(head, x);
}
Node* curr=head;
while(curr→next && curr→next→data!=K)
    { curr = curr→next;
    }
```

// out aya mtlb ya to prev pe pahuch gaya ord.

```
if(curr→next)
{
 Node* new = new node(x)
 newnode→next = curr→next;
 curr→next = newnode;
}
```

at a specific position (1-based indexig)

```
{ if (head == NULL) retu NULL;
  if (pos == 1) insert athead (head, x);
  Node* curr = head;
  for (int i=0; i < pos-1; i++)
      {
          curr = curr->next;
      }
  if (curr == NULL) return head;
  Node new = new Node*(x);
  newnode->next = curr => next;
  curr->next = newnode;
  return head;
}
```

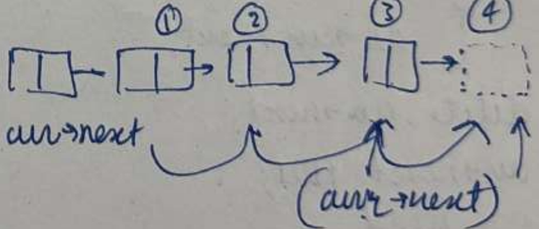


4-1 = (3) tk iterate kara

```
[ newnode->next = curr->next
[ curr->next = newnode,
```

At end

```
Node* newNode = new Node(val);
if (head == NULL)
    return newnode;
Node* curr = head;
while (curr->next)
    {
        curr = curr->next;
    }
curr->next = newnode;
return head.
```

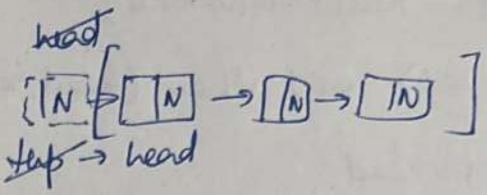


Cheatsheet

front = O(1)
After node = O(1)
beforenode = O(n)
atpos (O(n)
End = O(n)

Deletion:

① from the head; (O(1))



```
if(head == NULL) return NULL;
 Node* temp = head;
  head = temp→next;   // Slightly move head
  delete temp;
  return head;
}
```

② Delete from END:

```
if(head == NULL) return NULL;
if(head → next == NULL)
   {
      delete head;
      return NULL;
   }
Node* curr = head;
while(curr→~~head~~next→next)
   {
      curr = curr→next;
   }
 delete curr→next;
 curr→next = NULL;
 return head;
}
```

// for end,
→ iterate till while(curr→next→next)
// this way you can reach to end;
→ delete curr→next;
→ curr→next = NULL;
→ return head;

③ at Specific POS: (pos = 1 based indexing)

```
if(head == NULL) return NULL;
if(POS == 1) deletehead();

Node* curr = head;
~~w~~for(int i=1; i<pos-1, ++ curr, i++)
 {
  curr = curr→next;
 }
```

```
if(curr = NULL && curr→next = NULL)
   return head;
else
   Node* temp = curr→next
   curr→next = temp→next;
   delete temp;
return head.
```

Delete → move mt karo
No delete → tab move karo

① head delete hoga yani.
hamesha while no(if):

```
while (head && head → val = =val)
```

② middle / Enddelete

```
while( curr && curr→next)
```

// check for curr
curr→next (existance)
→ accessing (curr→next→val) good practise

③ ifelse

```
if (curr→next →val)
        // deletion
else // move.
```

Palindromic Linked list:

★ ①→②→③→②→①. here reading from front & end gives same result:

① go to mid using slow & fast pointer
② Reverse the list half way get the reversed list head
③ go to the head A & head B
→ traverse and check for vals, if diff (equal not) return false;

```
}
# if(head==NULL) returnNULL;
  L*s=head; L*f=head;
  while( f&&f→next)
  { s = s→next;
    f=f→next→next;
  }
L* heada = head;
L* headb = reverse(s);
while(heada && headb)
{ if (head→val != headb→val) return false;
  heada=heada→next;   headb=headb→next;
}
return true; }
```

```
→ Reverse (ListNode* s)
  { L* prev =NULL;
    L* curr = head;
    L* next=NULL;
    while(curr)
    {
      next=curr→next;
      curr→next= prev;
      prev=curr; curr=next;
    } return prev;
  };
```