BASICS:-

Pointers of arrays, strings and structures gives the address of first element or member.

Structures:

It is a user-defined datatype.

Name of Structure starts with a capital.

Structure concept came to overcome the array.

Typedef is used create alias names.

Declaration of a structure:

```
 struct  Student{
        int total;
        float per;
        char name[100] ;
}
```

Creating Alias Names for structure:

```
    typedef struct Student STD;
```

Initialization of variables:

STD s1;

Disadvantages of function :-

Returns only one element at a time.

To overcome this we use the concept of arrays and structures.

Stack , Queue and Linked Lists are Linear data structures.

Trees and Graphs are non-linear data structures.

Linked Lists:- 4 types

Single LL

Circular Singular LL

Double LL

Circular Double LL

Linked List is collection of nodes.

Node is divided into two parts.

data	Address of next node

In Linked Lists, address part of last node is always NULL which indicates the end of the linked lists.

Node Structure:

```
  struct  Node{
      int data;
      struct Node   *next;
};
      typedef  struct Node NODE;
      NODE     n1,n2,n3;
```

Linked Lists:-

When Linked List is empty. Both Head and Tail are at NULL.

Linked List -> empty => head =NULL and tail = NULL

Traversing means travelling from first node to last node.

1000	
10	2000
Head	

2000	
20	3000

3000	
30	4000

4000	
40	NULL
Tail	

Basic operations of Linked List:-:-

Algorithms of Insert , Delete and Display

General terms in all algorithms:-
NN = NewNode
res = result or value

Basic Operations in LL:

Insert
Delete
Display

Insert :-

Empty LL

When LL is empty, copy NN to head and tail i.e;
head = NN;
tail = NN

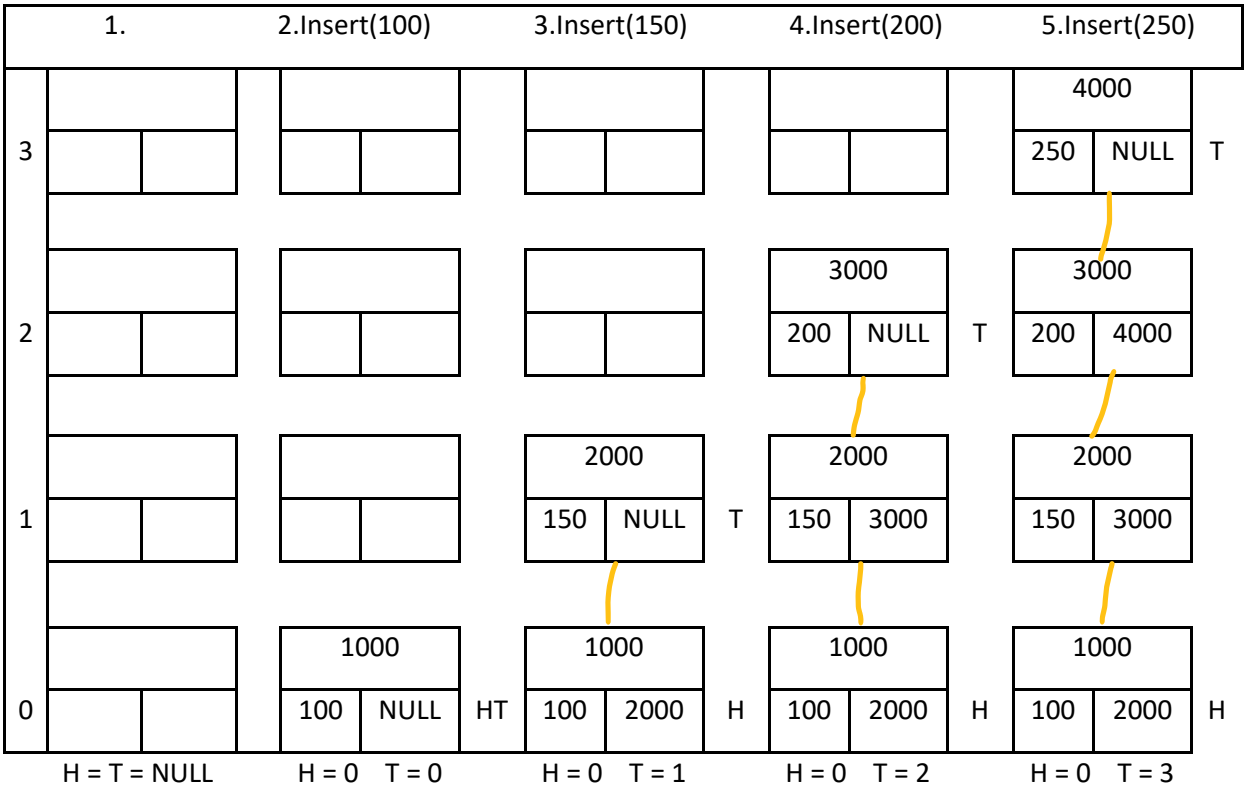
General condition

Copy NN to tail->next i.e;
tail->next = NN and copy NN to tail i.e; **tail = NN**

In the below table these are the steps for inserting different nodes :-

1. Since there are no nodes in a Linked List, Both head and tail are at NULL which indicates LL is empty.
2. Here we have inserted 100 with address 1000. Since there is no any other next node for 100. It's next node address is taken as NULL and there are no any link to it.
3. Here we have inserted 150 with address 2000.And next node's address of 100 as changed to 2000 to create a link between 100 and 150. Since there no other next node for 200. It's next node address is taken as NULL.

4 & 5 steps follows same as the 3rd step.



Delete :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data

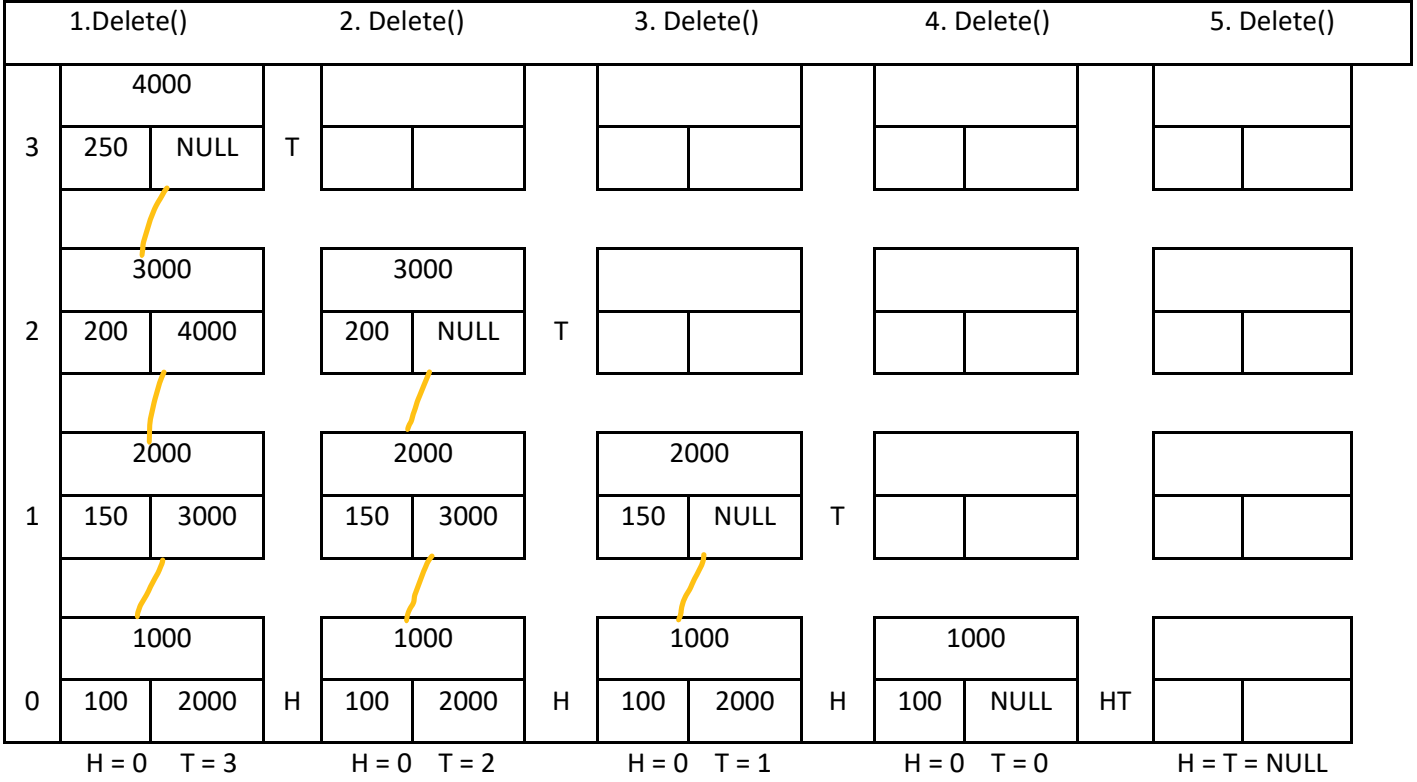
Empty LL	At First Node	General Condition
When LL is empty return **NULL**	If Head and Tail are equal (i.e; Both are at first node) Copy head to res . **res = head** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	copy head to temp i.e; **temp=head**. Using while loop for condition **temp->next->next** copy temp->next is copied to temp **temp = temp->next** and the loop continues. Then copy tail to res **res=tail** , NULL to temp->next **temp->next = NULL** and finally return **res**

In the below table these are the steps for deleting different nodes :-

In 1,2 & 3 Steps, last most step is deleted from a linked list and it's link with it's previous one is deleted and it's previous nodes next address is made as NULL and tail is moved to it's previous nodes.

4. Here we have deleted 150 with address 2000. Since there is no any other next node for 100 and 100 is the single node in the linked list . It's next node address is taken as NULL and there are no any link to it.
5. Since Head and Tail are at 0,After deleting node of 100, head and tail will be at NULL which indicates that the linked list is empty.

Now if we try to delete the nodes it returns NULL and displays No nodes.



Display :-

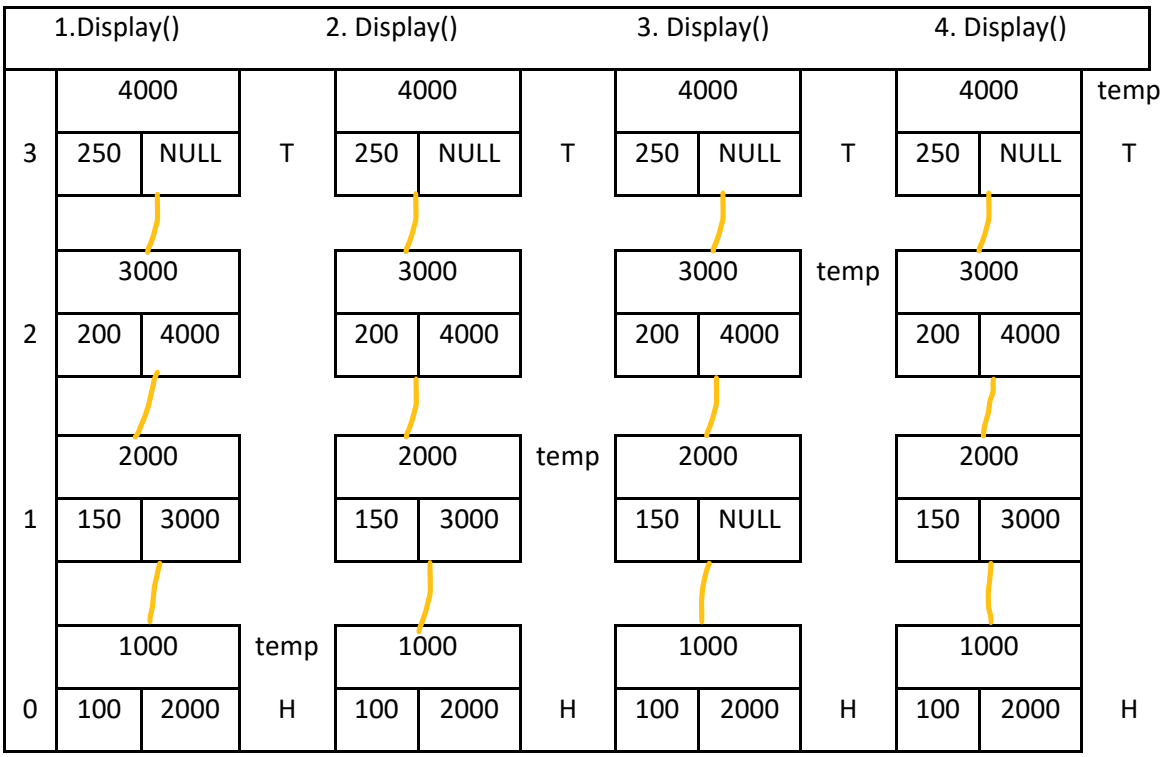
It displays nodes from head node to tail node by using the temp variable.

It displays the node which is at the position of temp.

When temp reaches tail node while loop stops Since tail->next == NULL.

If the linked list is empty, i.e; Head and tail are at NULL then it displays no nodes.

copy head to temp i.e; **temp=head**
Using while loop for condition **temp**
print temp and its data and next, then
copy temp->next is copied to temp
temp = temp->next and the loop
continues.



Output :- **100** **150** **200** **250**

Single Linked List:-

Algorithms of insert at tail, delete at tail, insert at head, delete at tail, Insert by position, delete by position, reverse and display.

Operations in SLL:
Insert at tail
Delete at tail
Insert at head
Delete at head
Insert by Position
Delete by Position
Display
reverse

Operations in SLL:
Insert at tail -> Same as insert in LL
Delete at tail -> Same as delete in LL
Display -> Same as display in LL

Display

copy head to temp i.e; **temp=head**
Using while loop for condition **temp**
print temp and its data & next,then
copy temp->next is copied to temp
temp = temp->next and the loop continues.

It displays nodes from head to tail in any case.

Insert at Tail :-

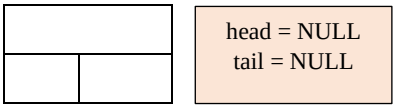
Empty LL

When LL is empty, copy NN to head and tail i.e;
head = NN;
tail = NN

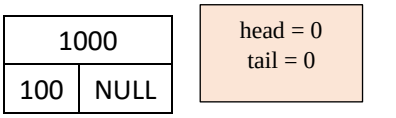
General condition

Copy NN to tail->next i.e;
tail->next = NN and copy NN to tail i.e; **tail = NN**

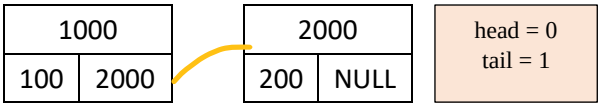
Since there is no any data in the node, Linked List is empty i.e; Head and tail are at NULL.



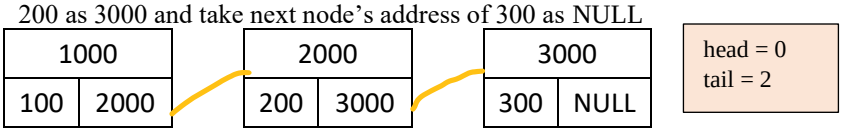
Insert(100) :- Since this is the First node and there are no any next nodes to it. It's next node's address is taken as NULL.



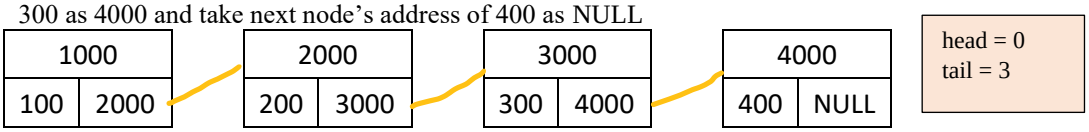
Insert(200) :- Here we are inserting the 200 with address 2000 which will be linked to 100 Node by changing the next node of 100 as 2000 and take next node's address of 200 as NULL.



Insert(300) :- Here we are inserting the 300 with address 3000 which will be linked to 200 Node by changing the next node of 200 as 3000 and take next node's address of 300 as NULL



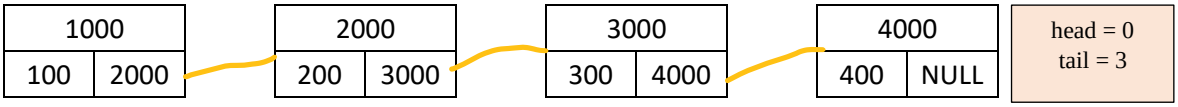
Insert(400) :- Here we are inserting the 400 with address 4000 which will be linked to 300 Node by changing the next node of 300 as 4000 and take next node's address of 400 as NULL



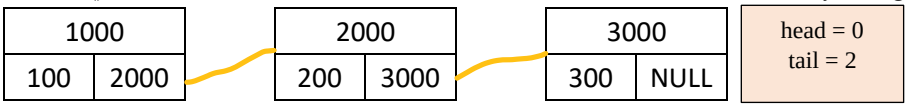
Delete at tail :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data.

Empty LL	**At First Node**	**General Condition**
When LL is empty return **NULL** to print **no nodes.**	If Head and Tail are equal (i.e; Both are at first node) Copy head to res . **res = head** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	copy head to temp i.e; **temp = head**. Using while loop for condition **temp->next->next** temp->next is copied to temp **temp = temp->next** and the loop continues. Then copy tail to res **res = tail**, NULL to temp->next **temp->next = NULL** and copy temp to tail i.e; **tail = temp** and finally return **res**

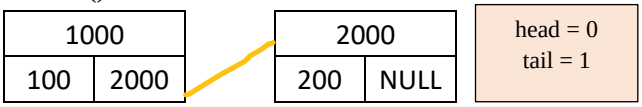
Delete() :- It deletes the lastmost node 400 and removes its link with 300 by changing the next node's address of 300 as NULL.



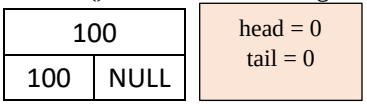
Delete() :- It deletes the lastmost node 300 and removes its link with 200 by changing the next node's address of 200 as NULL.



Delete() :- It deletes the lastmost node 200 and removes its link with 100 by changing the next node's address of 100 as NULL

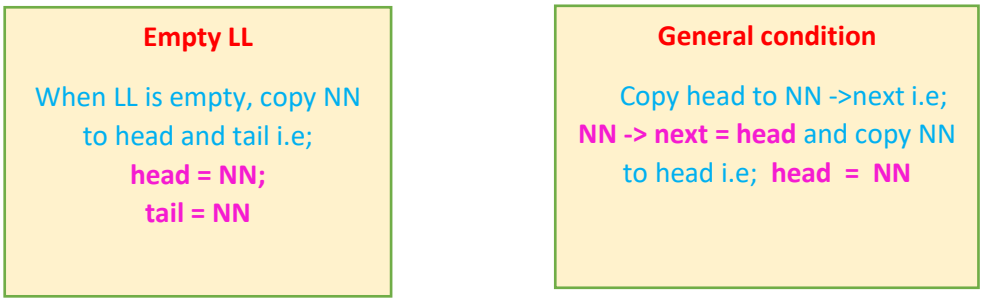


Delete() :- It deletes the single node 100 and changes Head and Tail to NULL

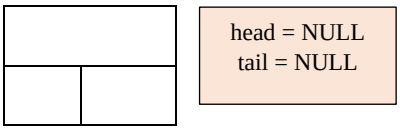


Now if we try to do delete operation it prints No nodes. Since LL is empty.

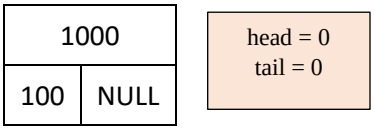
Insert at Head :-



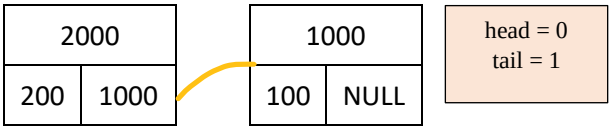
Since there is no any data in the node, Linked List is empty i.e; Head and tail are at NULL.



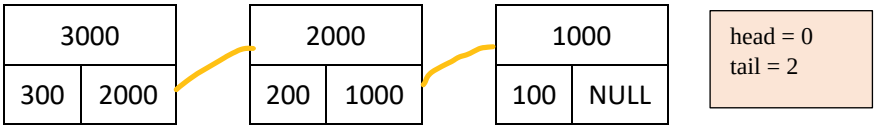
Insert(100) :- Since this is the First node and there are no any next nodes to it. It's next node's address is taken as NULL.



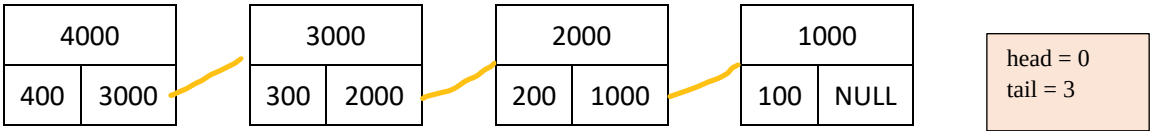
Insert(200) :- Here we are inserting the 200 with address 2000 which will be linked to 100 Node by changing the next node of 200 as 1000.



Insert(300) :- Here we are inserting the 300 with address 3000 which will be linked to 200 Node by changing the next node of 300 as 2000.



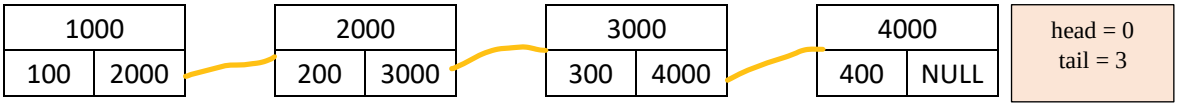
Insert(400) :- Here we are inserting the 400 with address 4000 which will be linked to 300 Node by changing the next node of 400 as 3000 and take next node's address of 100 as NULL



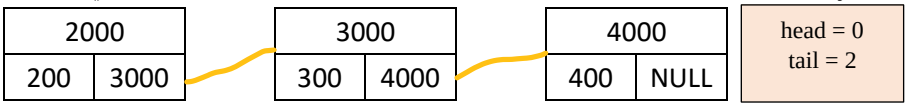
Delete at head :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data

Empty LL	At First Node	General Condition
When LL is empty return **NULL**	If Head and Tail are equal (i.e; Both are at first node). Copy head to res . **res = head** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	copy head to temp i.e; **temp=head**. Then copy head->next to head **head = head->next** , NULL to temp->next **temp->next = NULL** and finally return **temp**

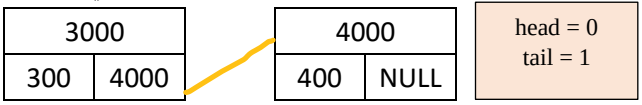
Delete() :- It deletes the frontmost node 100 and removes its link with 200 by changing the next node's address of 100 as NULL.



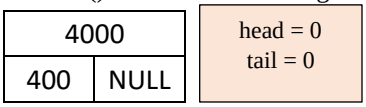
Delete() :- It deletes the frontmost node 200 and removes its link with 300 by changing the next node's address of 200 as NULL.



Delete() :- It deletes the frontmost node 300 and removes its link with 400 by changing the next node's address of 300 as NULL



Delete() :- It deletes the single node 400 and changes Head and Tail to NULL



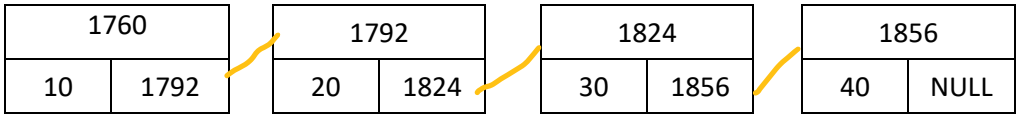
Now if we try to do delete operation it prints No nodes. Since LL is empty.

Reverse :- In this we have to declare some more pointer variables like cur, previous and next.

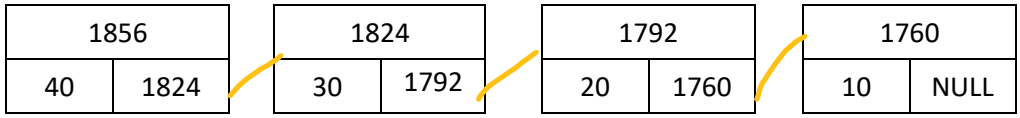
And Equate previous and next to NULL. [here cur=curnode = current node]

Empty LL	At First Node	General Condition
When LL is empty Print **No nodes**	If Head and Tail are equal (i.e; Both are at first node) Print **No need since list contains single node**.	copy head to tail and curnode i.e; **tail = head, cur = head** . Then while loop for condition **cur** and copy cur->next to next **next = cur->next** , copy prev to cur->next **cur->next = prev** , equate cur to prev **prev = cur** and equate next to cur **cur = next** and loop terminates when cur becomes null. Then copy prev to head **head = prev**

Suppose this is the Linked list before reversing.



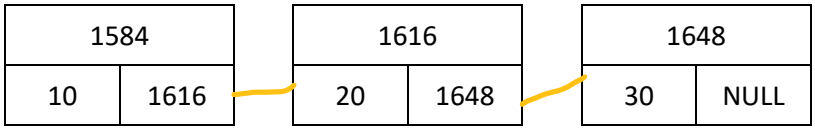
After reversing it will be changed as follows.



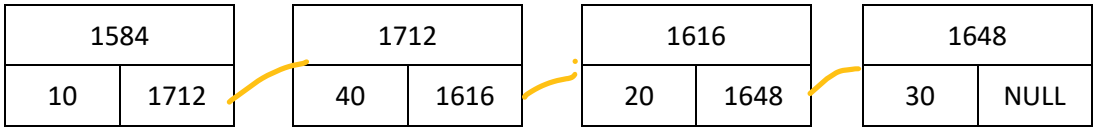
Insert by position :-

Empty LL	General Condition
When LL is empty, copy **NN** to head and tail i.e; **head = NN;** **tail = NN**	Copy head to temp and run the loop between **p = 1 to pos-1** by checking the condition **temp==NULL** if condition becomes **true print Insertion is not possible** and equate flag to 1 **flag = 1** and break it and copy temp->next to temp **temp = temp->next** and terminate the loop. If **flag ==** 0 copy temp->next to NN->next i.e; **NN->next = temp-next** and NN to temp->next i.e; **Temp->next = NN.**

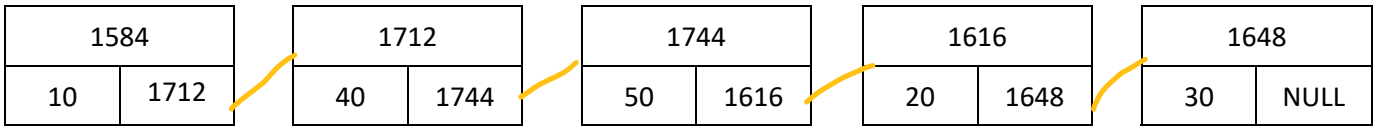
Suppose this is our Linked List



If we insert 40 at 1st position the changed linked list will be like this



If we insert 50 at 2nd position the changed linked list will be like this

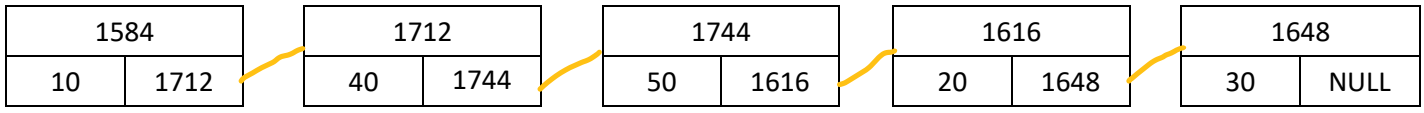


If we try insert at position greater than (size – 1)th position it prints **Insertion is not possible.**

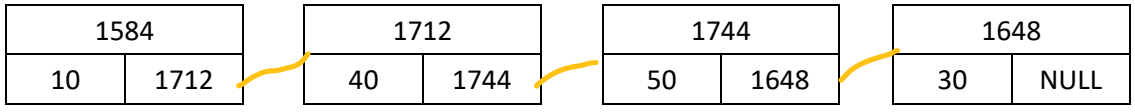
Delete by position :-

Empty LL	General Condition
When LL is empty return **NULL**	Copy head to temp and run the loop between **p = 1 to pos-1** by checking the condition **temp==NULL** if condition becomes **true return null** and copy temp->next to temp **temp = temp->next** and terminate the loop. Then copy temp->next to res i.e; **res = temp->next** , Temp->next->next to temp->next i.e; **temp->next = temp->next->next** and equate NULL to res->next i.e; **res->next = NULL** and return **res**

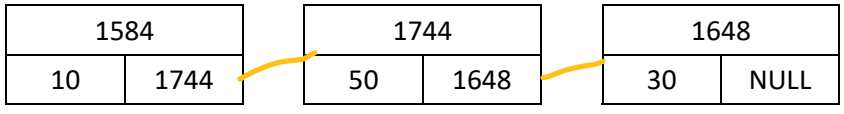
Suppose this is our Linked List



If we delete node at 3rd position the changed linked list will be like this



If we delete node at 1st position the changed linked list will be like this



If we try delete at position when the Linked list is empty it prints **Deletion is not possible or No nodes.**

Circular Linked List:-

Algorithms of insert at tail, delete at tail, insert at head, delete at tail, Insert by position, delete by position and display.

Operations in CLL:
Insert at tail
Delete at tail
Insert at head
Delete at head
Insert by Position
Delete by Position
Display
reverse

Display

copy head to temp i.e; **temp=head**
Using while loop for condition **temp**
print temp and its data & next,then
copy temp->next is copied to temp
temp = temp->next and the loop continues.

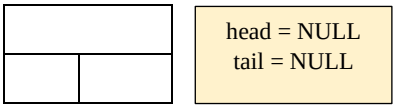
It displays nodes from head to tail in any case.

Insert at Tail :-

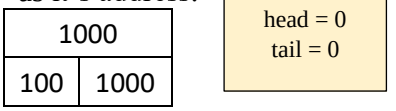
Empty LL
When LL is empty, copy NN to head and tail i.e;
head = NN;
tail = NN
copy head to head->next i.e;
head->next = head

General condition
Copy NN to tail->next i.e;
tail->next = NN and copy NN to tail i.e; **tail = NN** and copy head to tail->next i.e;
tail->next = head

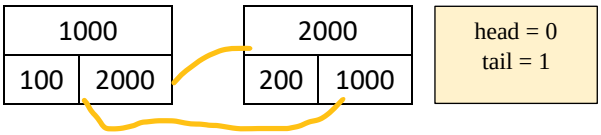
Since there is no any data in the node, Linked List is empty i.e; Head and tail are at NULL.



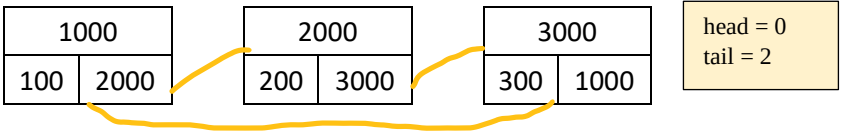
Insert(100) :- Since this is the First node and there are no any next nodes to it. It's next node's address is taken as it's address.



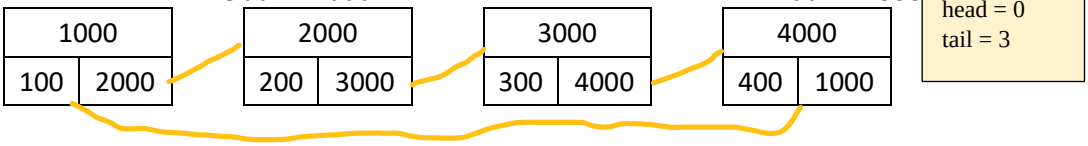
Insert(200) :- Here we are inserting the 200 with address 2000 which will be linked to 100 Node by changing the next node of 100 as 2000 and take next node's address of 200 as 1000.



Insert(300) :- Here we are inserting the 300 with address 3000 which will be linked to 200 Node by changing the next node of 200 as 3000 and take next node's address of 300 as 1000.



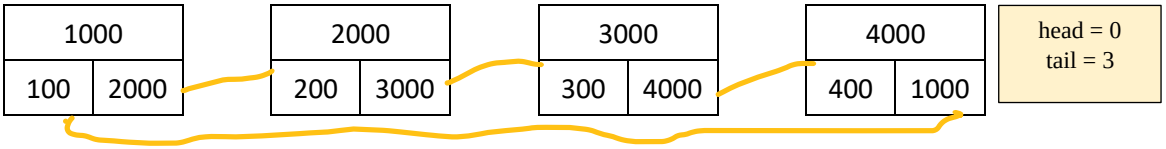
Insert(400) :- Here we are inserting the 400 with address 4000 which will be linked to 300 Node by changing the next node of 300 as 4000 and take next node's address of 400 as 1000



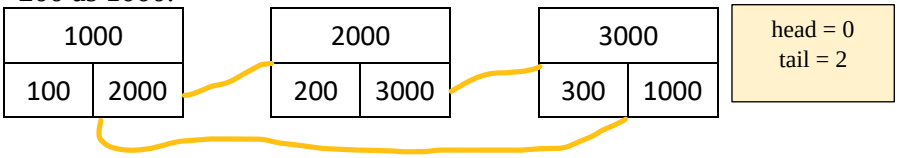
Delete at tail :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data.

Empty LL	At First Node	General Condition
When LL is empty return **NULL** to print **no nodes.**	If Head and Tail are equal (i.e; Both are at first node) Copy head to res i.e; **res = head.** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	copy head to temp i.e; **temp=head**. Using while loop for condition **temp->next->next != head** temp->next is copied to temp **temp = temp->next** and the loop continues. Then copy tail to res i.e; **res = tail**, head to temp->next i.e; **temp->next = head**, temp to tail i.e; **tail = temp** and NULL to res->next **res->next = NULL** and finally return **res**

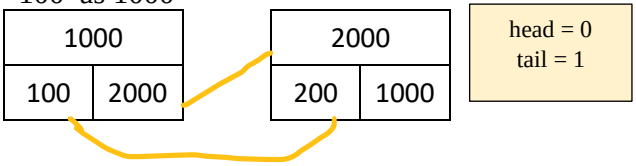
Delete() :- It deletes the lastmost node 400 and removes its link with 300 by changing the next node's address of 300 as 1000.



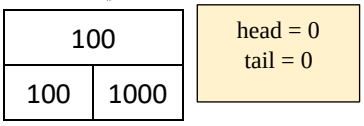
Delete() :- It deletes the lastmost node 300 and removes its link with 200 by changing the next node's address of 200 as 1000.



Delete() :- It deletes the lastmost node 200 and removes its link with 100 by changing the next node's address of 100 as 1000



Delete() :- It deletes the single node 100 and changes Head and Tail to NULL.

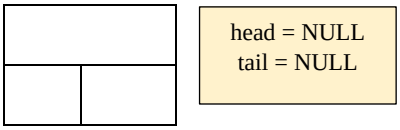


Now if we try to do delete operation it prints No nodes. Since LL is empty.

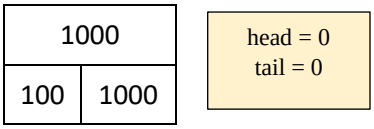
Insert at Head :-

Empty LL	General condition
When LL is empty, copy NN to head and tail i.e; **head = NN;** **tail = NN** copy head to head->next i.e; **head->next = head**	Copy head to NN ->next i.e; **NN -> next = head** and copy NN to head i.e; **head = NN** and copy head to tail->next i.e; **tail->next = head**

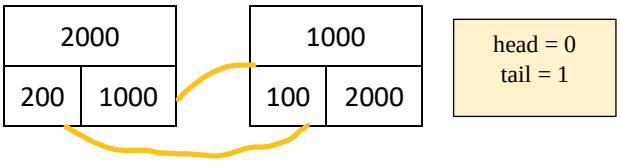
Since there is no any data in the node, Linked List is empty i.e; Head and tail are at NULL.



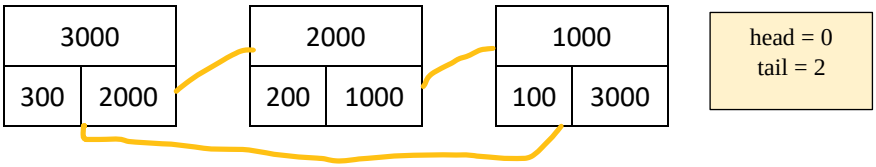
Insert(100) :- Since this is the First node and there are no any next nodes to it. It's next node's address is taken as 1000.



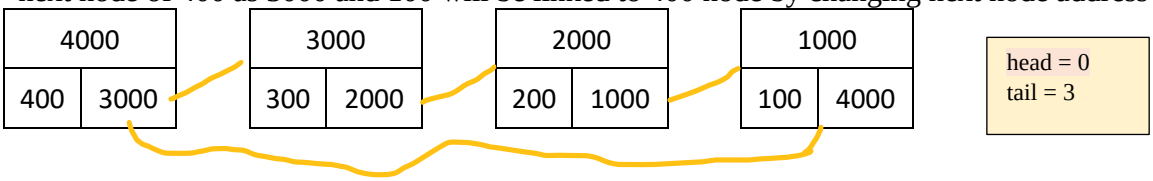
Insert(200) :- Here we are inserting the 200 with address 2000 which will be linked to 100 Node by changing the next node address of 200 as 1000 and 100 will be linked to 200 node by changing next node address of 100 as 2000



Insert(300) :- Here we are inserting the 300 with address 3000 which will be linked to 200 Node by changing the next node address of 300 as 2000 and 100 will be linked to 300 node by changing next node address of 100 as 3000



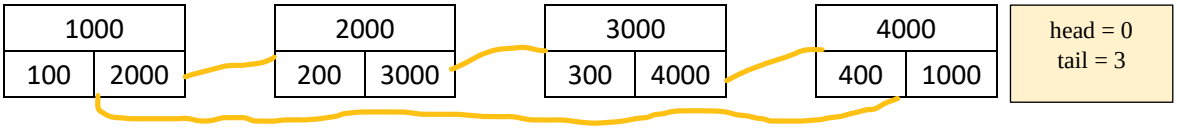
Insert(400) :- Here we are inserting the 400 with address 4000 which will be linked to 300 Node by changing the next node of 400 as 3000 and 100 will be linked to 400 node by changing next node address of 100 as 4000



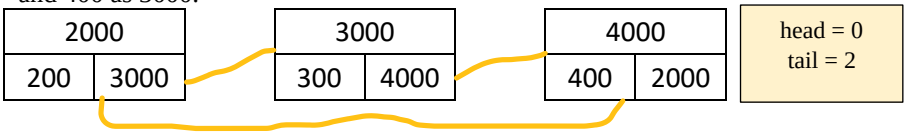
Delete at head :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data

Empty LL When LL is empty return **NULL**	**At First Node** If Head and Tail are equal (i.e; Both are at first node). Copy head to res . **res = head** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	**General Condition** copy head to res i.e; **res = head**. Then copy head->next to head i.e; **head = head->next** , copy head to tail->next i.e; **tail->next = head** NULL to res->next i.e; **res->next = NULL** and finally return **res.**

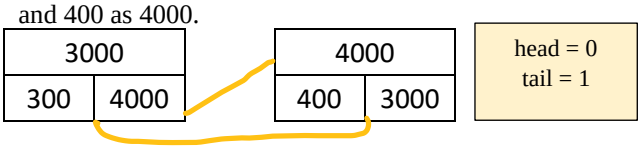
Delete() :- It deletes the frontmost node 100 and removes its link with 200 by changing the next node's address of 100 as NULL and 400 as 2000.



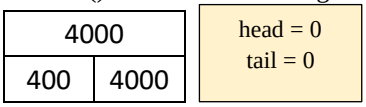
Delete() :- It deletes the frontmost node 200 and removes its link with 300 by changing the next node's address of 200 as NULL and 400 as 3000.



Delete() :- It deletes the frontmost node 300 and removes its link with 400 by changing the next node's address of 300 as NULL and 400 as 4000.



Delete() :- It deletes the single node 400 and changes Head and Tail to NULL

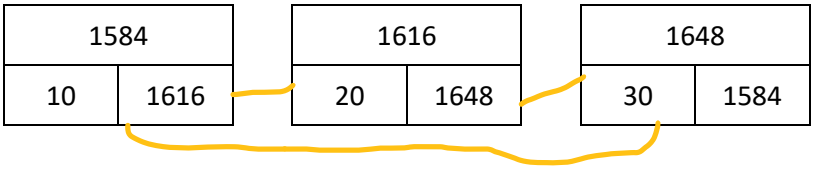


Now if we try to do delete operation it prints No nodes. Since LL is empty.

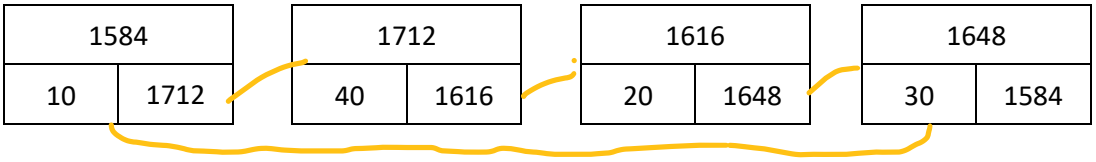
Insert by position :-

Empty LL	**General Condition**
When LL is empty, copy **NN** to head and tail i.e; **head = NN;** **tail = NN** copy head to head->next i.e; **head->next = head**	Copy head to temp and run the loop between **p = 1 to pos-1** by checking the condition **temp->next == head** if condition becomes **true print Insertion is not possible** and return it otherwise copy temp->next to temp **temp = temp->next** and terminate the loop. Then copy temp->next to NN->next i.e; **NN->next = temp->next** and copy NN to temp -> next i.e; **temp->next = NN**

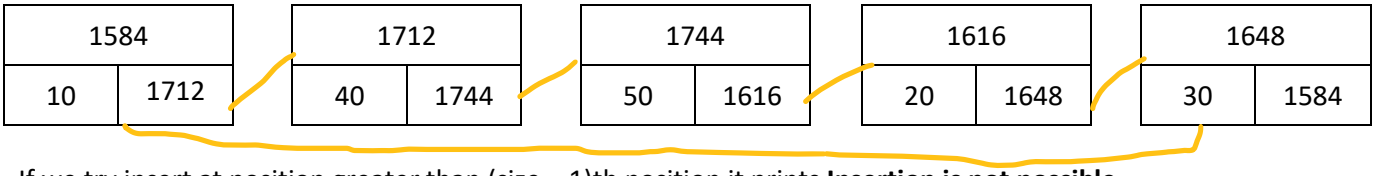
Suppose this is our Linked List



If we insert 40 at 1st position the changed linked list will be like this.



If we insert 50 at 2nd position the changed linked list will be like this

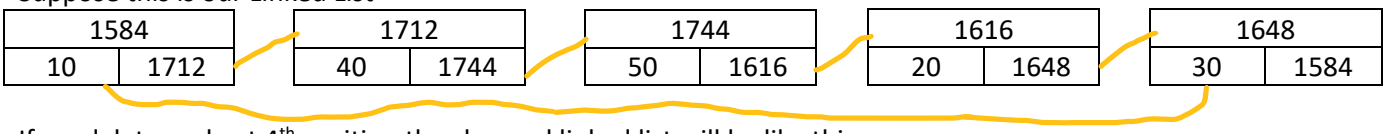


If we try insert at position greater than (size – 1)th position it prints **Insertion is not possible.**

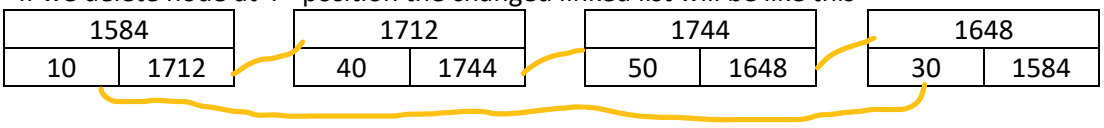
Delete by position :-

Empty LL	**General Condition**
When LL is empty return **NULL**	Copy head to temp and run the loop between **p = 1 to pos-1** by checking the condition **temp->next == head** if condition becomes **true** print **'Deletion is not possible'** return **null** and copy temp->next to temp **temp = temp->next** and terminate the loop. Then copy temp->next to res i.e; **res = temp->next** , res->next to temp->next i.e; **temp->next = res->next** and equate NULL to res->next i.e; **res->next = NULL** and return **res**

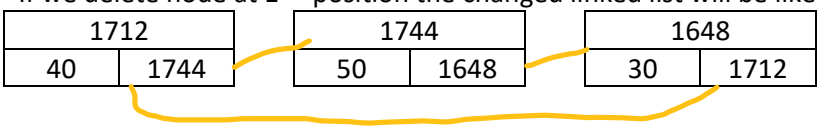
Suppose this is our Linked List



If we delete node at 4th position the changed linked list will be like this



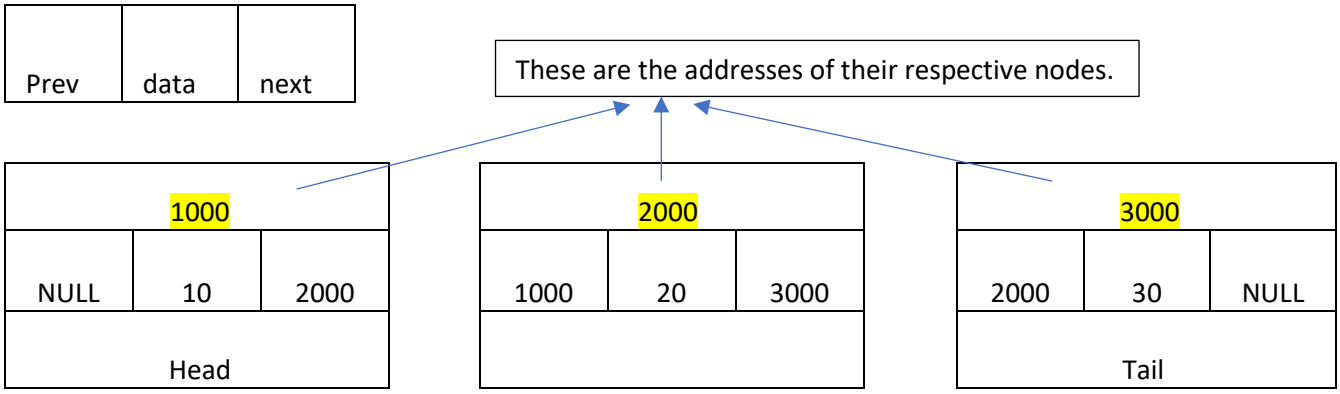
If we delete node at 1st position the changed linked list will be like this



If we try delete at position when the Linked list is empty it prints **Deletion is not possible or No nodes.**

Double Linked List:-

It consists of three nodes. 1st node contains address of previous node , middle node contains data of that node and the last node contains address of next node



Algorithms of insert at tail, delete at tail, insert at head, delete at tail, Insert by position, delete by position, reverse and display.

Operations in SLL:

Insert at tail
Delete at tail
Insert at head
Delete at head
Insert by Position
Delete by Position
Display from head to tail
Display from tail to head

Insert at Tail :-

Empty LL

When LL is empty, copy NN to head and tail i.e;
head = NN;
tail = NN

General condition

Copy NN to tail->next i.e; **tail->next = NN**, copy tail to NN-> prev i.e; **NN->prev = tail** and copy NN to tail i.e;
tail = NN

Since there is no any data in the node, Linked List is empty i.e; Head and tail are at NULL.

Insert(100) :- Since this is the First node and there are no any next nodes to it. It's next and previous node's address is taken as NULL.

1000		
NULL	100	NULL

Insert(200) :- Here we are inserting the 200 with address 2000 which will be linked to 100 Node by changing the next node of 100 as 2000 and take previous node's address of 200 as 1000.

1000		
NULL	100	2000

2000		
1000	200	NULL

Insert(300) :- Here we are inserting the 300 with address 3000 which will be linked to 200 Node by changing the next node of 200 as 3000 and take previous node's address of 300 as 2000.

1000		
NULL	100	2000

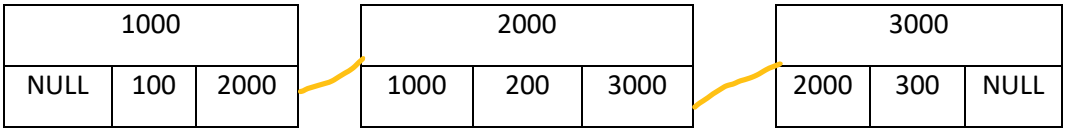
2000		
1000	200	3000

3000		
2000	300	NULL

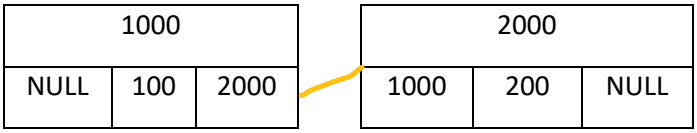
Delete at tail :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data.

Empty LL	**At First Node**	**General Condition**
When LL is empty return **NULL** to print **no nodes.**	If Head and Tail are equal (i.e; Both are at first node) Copy head to res . **res = head** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	copy tail to res i.e; **res=tail** , tail->prev to tail i.e; **tail = tail->prev** copy NULL to tail->Next and res->prev i.e; **tail -> Next = NULL** **res -> prev = NULL** and finally return **res**

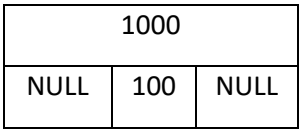
Delete() :- It deletes the lastmost node 300 and removes its link with 200 by changing the next node's address of 200 and previous nodes address of 300 as NULL.



Delete() :- It deletes the lastmost node 200 and removes its link with 100 by changing the next node's address of 100 and previous nodes address of 200 as NULL.



Delete() :- It deletes the single node 100 and changes Head and Tail to NULL

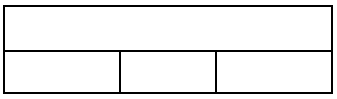


Now if we try to do delete operation it prints No nodes. Since LL is empty.

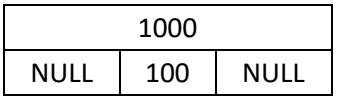
Insert at Head :-

Empty LL	**General condition**
When LL is empty, copy NN to head and tail i.e; **head = NN;** **tail = NN**	Copy head to NN ->next i.e; **NN -> next = head**, copy NN to head->prev i.e; **head->prev = NN** and copy NN to head i.e; **head = NN**

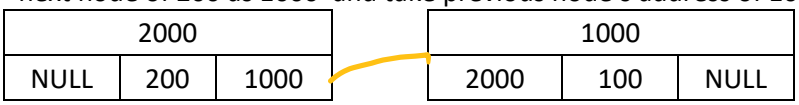
Since there is no any data in the node, Linked List is empty i.e; Head and tail are at NULL.



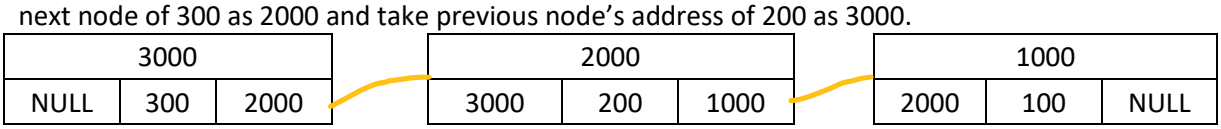
Insert(100) :- Since this is the First node and there are no any next nodes to it. It's next and previous node's address is taken as NULL.



Insert(200) :- Here we are inserting the 200 with address 2000 which will be linked to 100 Node by changing the next node of 200 as 1000 and take previous node's address of 100 as 2000.



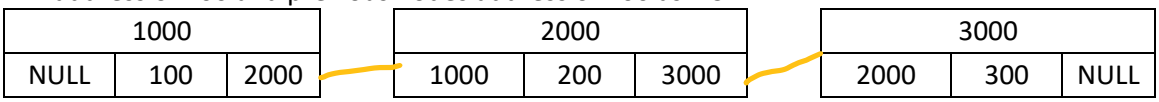
Insert(300) :- Here we are inserting the 300 with address 3000 which will be linked to 200 Node by changing the next node of 300 as 2000 and take previous node's address of 200 as 3000.



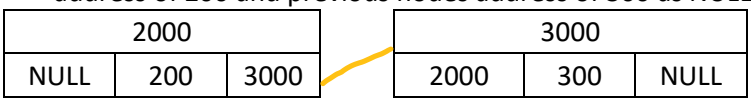
Delete at head :- If nodes are there we print the data of res and and apply free function to it. free function is used to remove memory of that data

Empty LL	At First Node	General Condition
When LL is empty return **NULL**	If Head and Tail are equal (i.e; Both are at first node). Copy head to res . **res = head** Then equate head and tail to NULL **[head = NULL; tail = NULL]** since the existing LL becomes empty and finally return **res**	copy head to res i.e; **res = head**. Then copy head->next to head i.e; **head = head->next** , NULL to head->prev and res->next i.e; **head->prev = NULL** **res->next = NULL** and finally return **res**

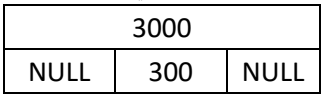
Delete() :- It deletes the frontmost node 100 and removes its link with 200 by changing the next node's address of 100 and previous nodes address of 200 as NULL.



Delete() :- It deletes the frontmost node 200 and removes its link with 300 by changing the next node's address of 200 and previous nodes address of 300 as NULL.



Delete() :- It deletes the single node 300 and changes Head and Tail to NULL

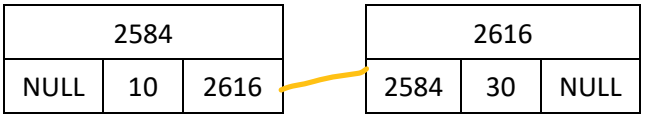


Now if we try to do delete operation it prints No nodes. Since LL is empty.

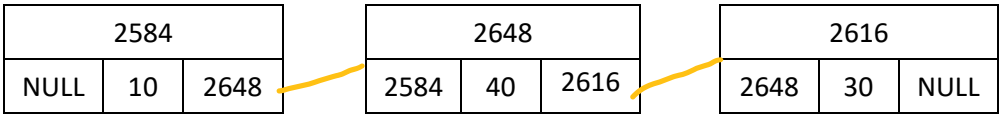
Insert by position :-

Empty LL	General Condition
When LL is empty, copy **NN** to head and tail i.e; **head = NN;** **tail = NN**	Copy head to temp i.e; **temp = head** and run the loop between **p = 1 to pos-1** by checking the condition **temp==NULL** if condition becomes **true print Insertion is not possible** and return it and copy temp->next to temp **temp = temp->next** and terminate the loop. Then copy temp->next to NN->next i.e; **NN->next = temp->next** , temp to NN->prev i.e; **NN->prev = temp** , also copy NN to temp->next i.e; **temp->next = NN** and NN to NN->next->prev i.e; **NN->next->prev = NN.**

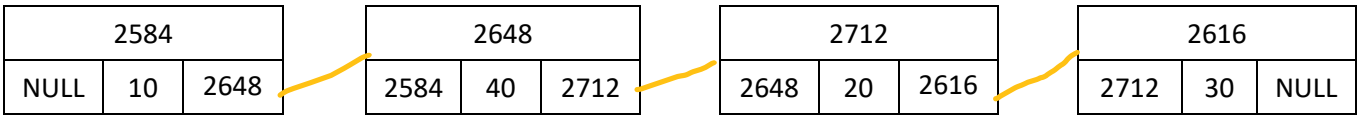
Suppose this is our Linked List



If we insert 40 at 1st position the changed linked list will be like this



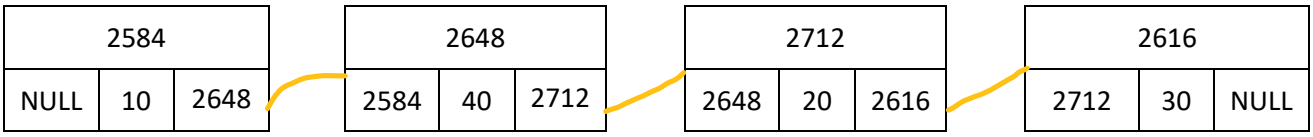
If we insert 20 at 2nd position the changed linked list will be like this



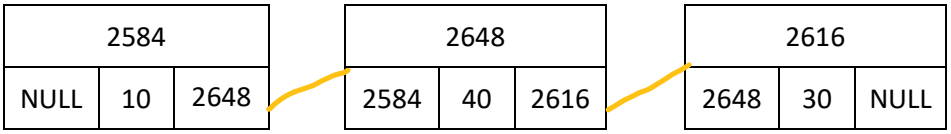
Delete by position :-

Empty LL When LL is empty return **NULL**	**General Condition** Copy head to temp and run the loop between **p = 1 to pos-1** by checking the condition **temp->next == head** if condition becomes **true** print **'Deletion is not possible' return null** and copy temp->next to temp **temp = temp->next** and terminate the loop. Then copy temp->next to res i.e; **res = temp->next** ,Temp->next->next to temp->next i.e; **temp->next = temp->next->next** , copy temp to res->prev i.e; **res->prev = temp** and equate NULL to res->prev and res->next i.e; **res->prev = NULL and res->next = NULL** and return **res**

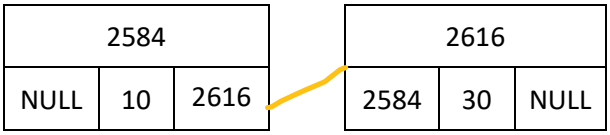
Suppose this is our Linked List



If we delete node at 2nd position the changed linked list will be like this



If we delete node at 1st position the changed linked list will be like this

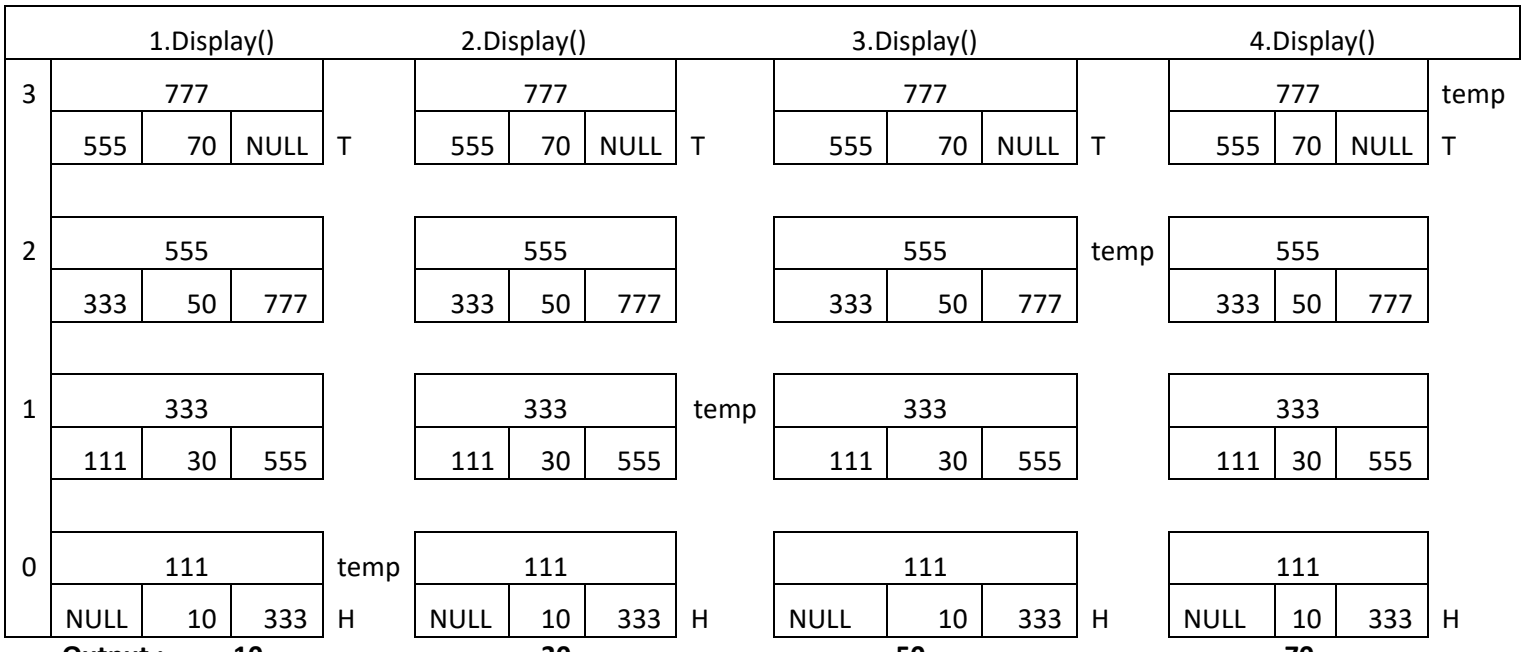


Display from head to tail :-

Empty LL

When LL is empty it prints **No nodes**

copy head to temp i.e; **temp =head** Using while loop for condition **temp** print data of temp then copy temp->next is copied to temp **temp = temp->next** and the loop continues.

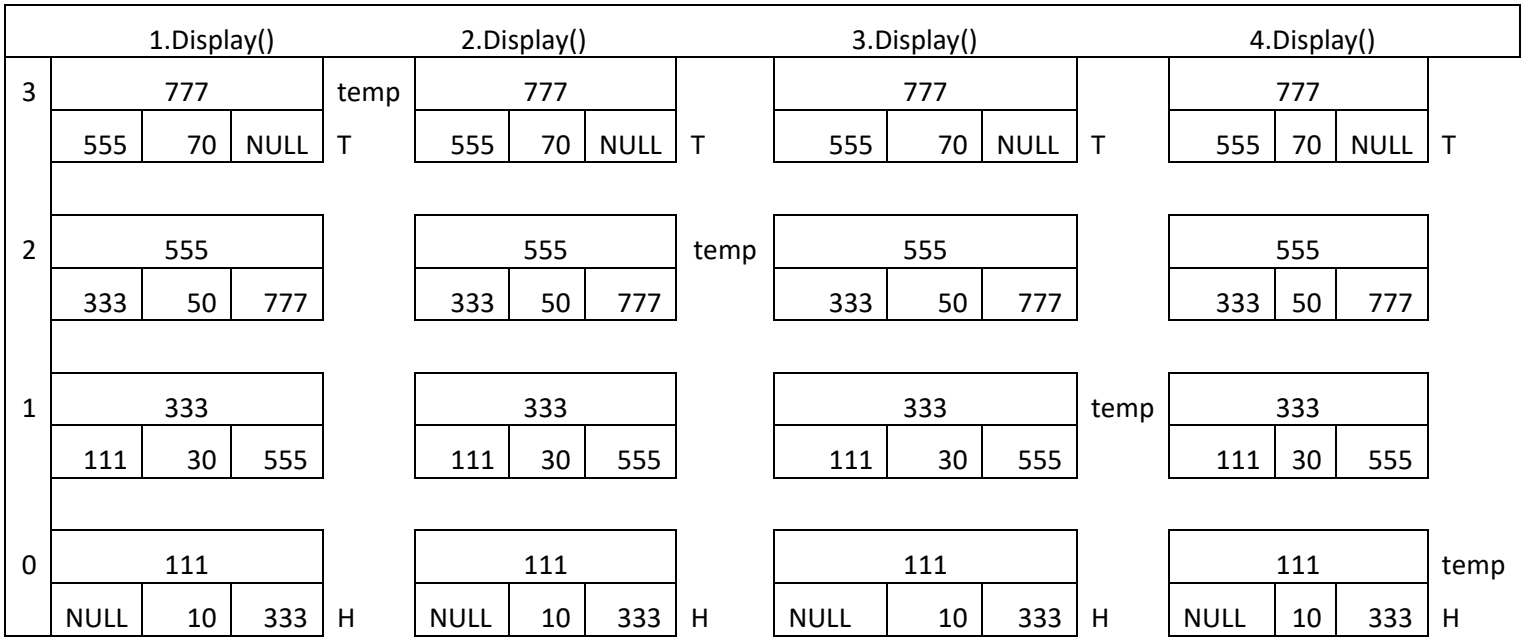


Display from tail to head :-

Empty LL

When LL is empty it prints **No nodes**

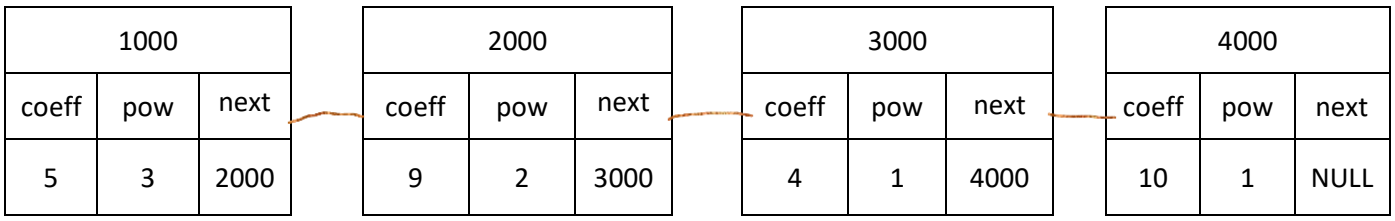
copy tail to temp i.e; **temp = tail** Using while loop for condition **temp** print data of temp then copy temp->prev is copied to temp **temp = temp->prev** and the loop continues.



Output : - **70** **50** **30** **10**

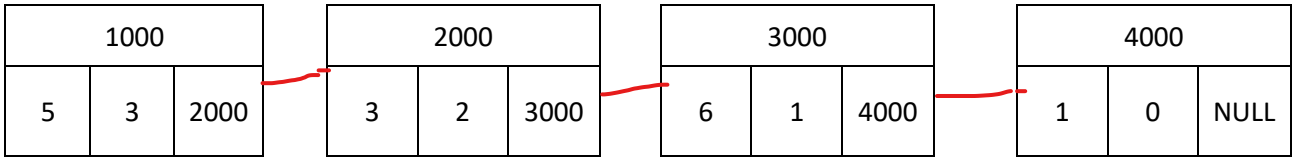
Polynomial Representation of Linked Lists :-

5x^3+9x^2+4x+10

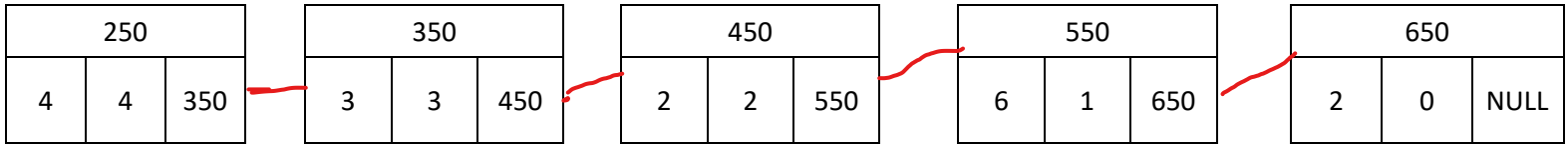


For two polynomials :-

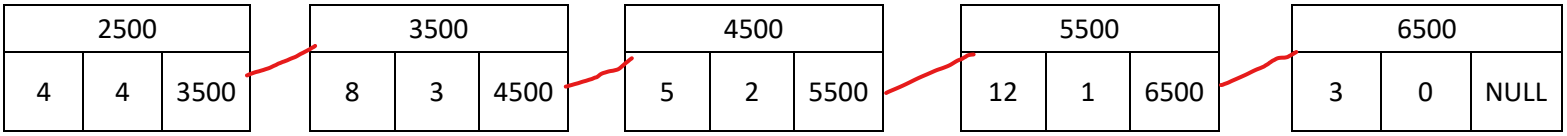
5x^3+3x^2+6x+1



4x^4 + 3x^3+2x^2+6x+2



Their resultant will be there sum



Sparks Matrix :-

C R	0	1	2	3	4
0					
1					
2					
3					
4