

Sunday Live → OOPS → Practical OOPS
↓
PW Skills

Linked List

Part – 1

User defined data type ✓

Raghav Garg

Revision of OOPS

Defining a class

Declaring and Initialising an object

COLLEGE
WALLAH

```
class Student {
```

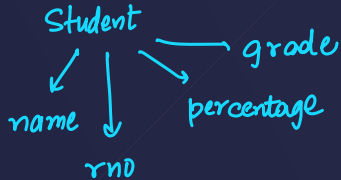
```
public:
```

```
    int rno;
```

```
    float percent;
```

```
    string name;
```

```
};
```



raghav → name

→ rno

→ percent



s

string name = "Raghav"

int rno = 76

float per = 92.6

```
void change(Student s){
    s.name = "Harsh";
}

int main(){
    Student s("Raghav",76,92.6);
    cout<<s.name<<endl;
    change(s);
    cout<<s.name<<endl;
}
```

"Raghav"	76	92.6
----------	----	------

name	roll	marks
"Raghav"	76	92.6

s

Output:

- Raghav

int x = 4;

change(x)

COLLEGE
WALLAH

Revision of OOPS

Constructors & making parameterised constructors

COLLEGE
WALLAH

Arrays

→ Linear D.S. → multiple dabbe bana sakte hai

Limitations



1) Fixed Size → **vector**

↓
problem

list

arr

0	1	2	3	4
100	200	400	300	0

arr[2] = 500; // O(1) T.C

vector<int> arr(5);

10	20	30	40	50
----	----	----	----	----

v. push_back(100)

10	20	30	40	50	100				
----	----	----	----	----	-----	--	--	--	--

Arrays

Limitations

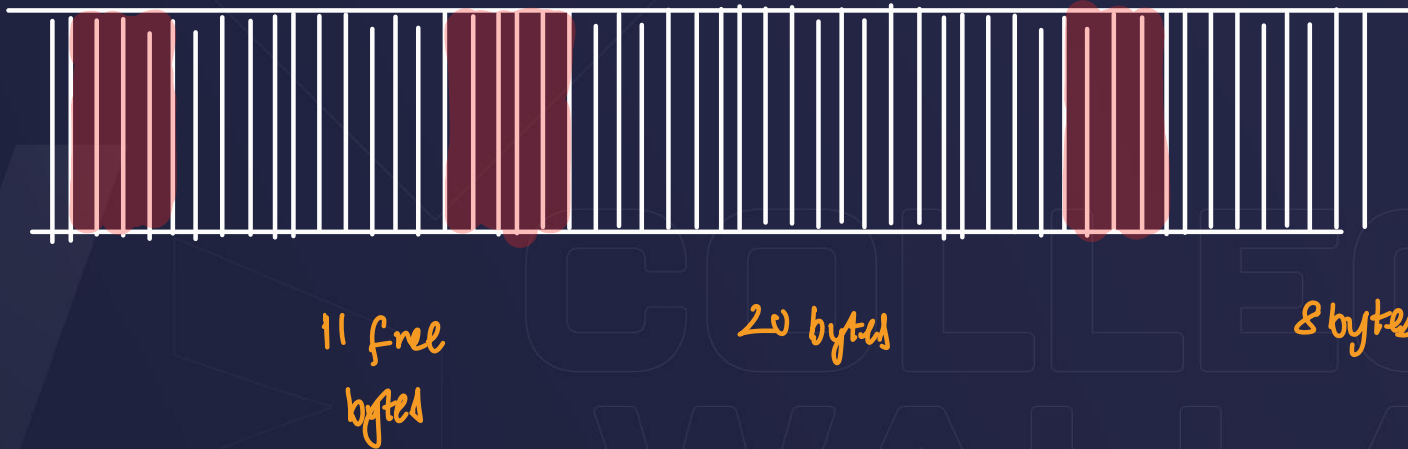
2) Contiguous memory allocation :

```
int arr[4];
```

↓

16 bytes

```
int arr[6]; → 6 × 4 = 24 bytes
```



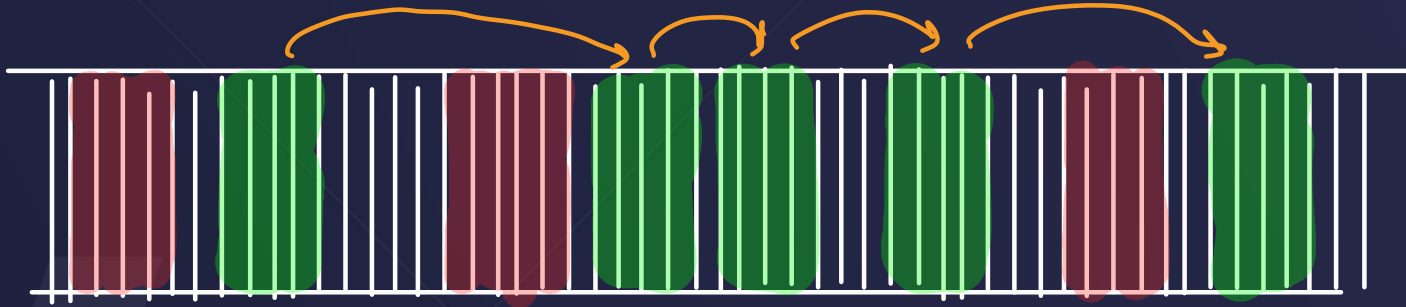
Arrays

Need for a new linear data structure

COLLEGE
WALLAH

Introduction to Linked List

Idea of linking two non-contiguous memory locations (nodes)



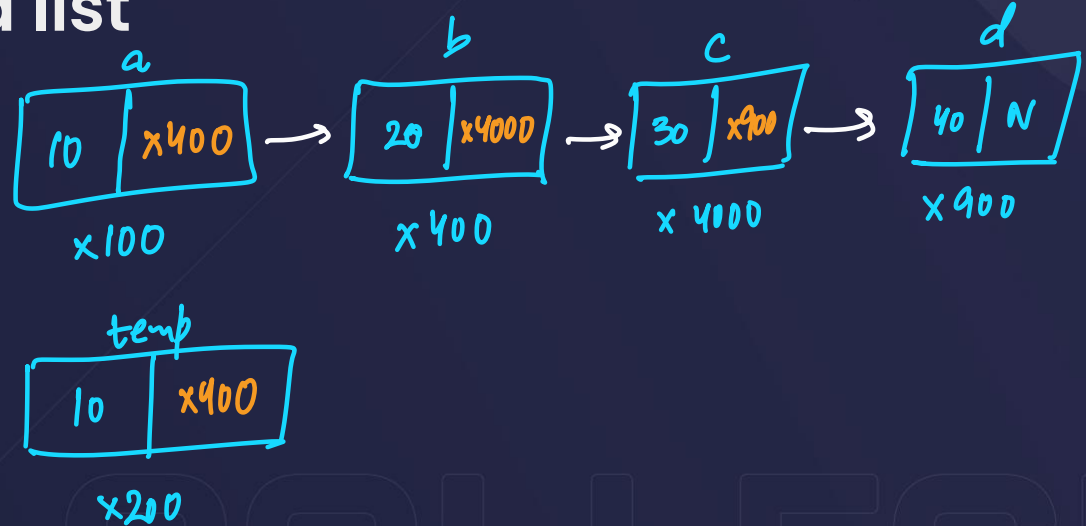
5 size ki lk list

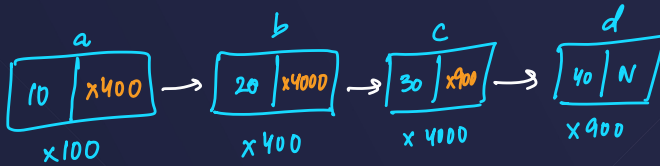
Introduction to Linked List

```
class Node{ // Linked List Node
public:
    int val;
    Node* next;
    Node(int val){
        this->val = val;
        this->next = NULL;
    }
};

int main(){
    // 10 20 30 40
    Node a(10);
    Node b(20);
    Node c(30);
    Node d(40);
    // forming ll
    a.next = &b;
    b.next = &c;
    c.next = &d;
}
```

ed list

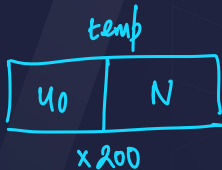




• 10 20 30

```

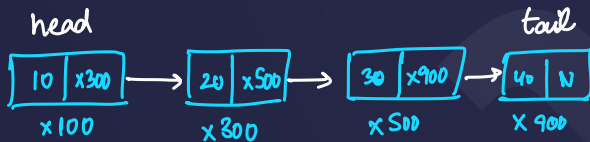
Node temp = a;
while(temp.next!=NULL){
    cout<<temp.val<<" ";
    temp = *(temp.next);
}
  
```



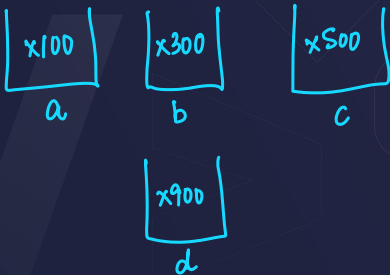
COLLEGE
WALLAH

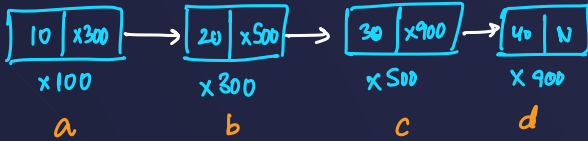
```

✓ Node* a = new Node(10);
✓ Node* b = new Node(20);
✓ Node* c = new Node(30);
✓ Node* d = new Node(40);
✓ a->next = b;
✓ b->next = c;
✓ c->next = d;
    
```



$a \rightarrow next$
 \downarrow
 $(*a).next$





NULL

temp

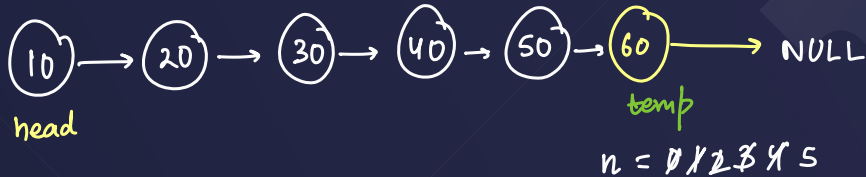
```

Node* temp = a;
while(temp!=NULL){
    cout<<temp->val<<" ";
    temp = temp->next;
}
  
```

Output

10 20 30 40

Representation :



`n = 0;`

`Node* temp = head`

`while (temp != null) {`

`temp = temp->next;`

`n++;`

`}`

Introduction to Linked List

Does linked list overcomes the limitations of arrays?

→ Unlimited Size

→ Continuous memory allocation

COLLEGE
WALLAH

Implementation

Node class

```
class Node{
    int val;
    Node* next;
};
```

With Parameterised constructor

```
class Node{
    int val;
    Node* next;
    Node(int val){
        this->val = val;
        this->next = NULL;
    }
};
```


Implementation

Linking nodes to form a linked list

COLLEGE
WALLAH

Displaying a Linked List ✓

2. size

```
Node* temp = head;
```

```
while (temp != NULL) {
```

```
    cout << temp->val;
```

```
    temp = temp->next;
```

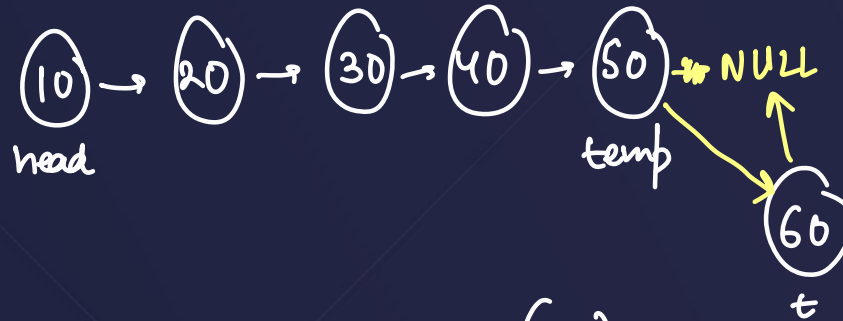
```
}
```

T.C. = $O(n)$

S.C. = $O(1)$

COLLEGE
WALLAH

Displaying a Linked List



```

while (temp->next != NULL)
|   temp = temp->next;
3
temp->next = t;
  
```

```
Node *t = new Node(60);
```

T.C. = $O(n)$

Displaying a Linked List

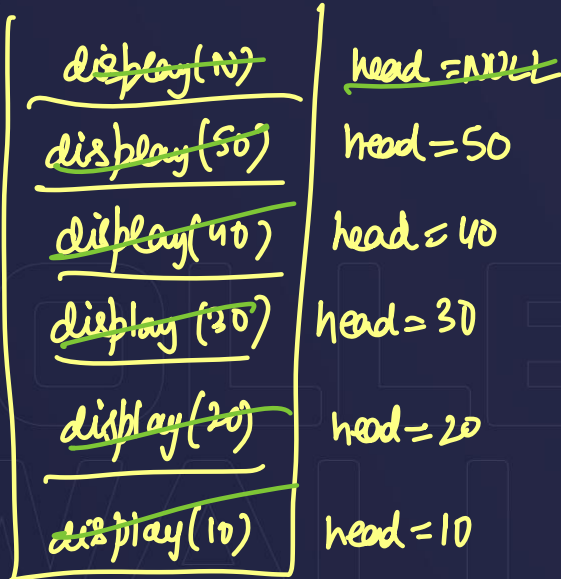
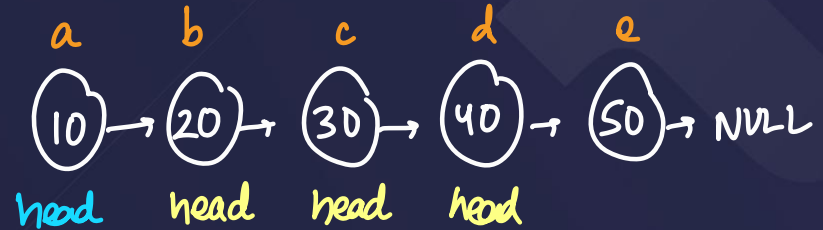
Can we do it recursively?

```
void display (Node* head) {
    if (head == NULL) return;
    cout << head->val;
    display (head->next);
}
```

3

Output

10 20 30 40 50



T.C. = $O(n)$

S.C. = $O(n)$

What will this function do?

```
void display(Node* head) {
    if(head == NULL) return;
    display(head->next);
    cout<<head->val<< " ";
}
```

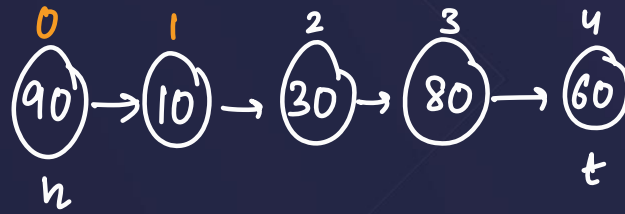
```
void display (Node* head){
    if(head == NULL) return;
    cout << head->val;
    display (head->next);
}
```

- (a) Print all the elements of the linked list.
- (b) Print all the elements except last one.
- (c) Print alternate nodes of linked list
- ✓ (d) Print all the nodes in reverse order

Implementation

Linked List class

```
class LinkedList {
    Node* head;
    Node* tail;
    int size = 0;
}
```



```
LinkedList ll;
ll.add(60);
ll.addAtHead(90);
ll.deleteAt(2);
ll.insertAt(3, 80);
```



size = 0

Display method

Implement display method to print all the elements

COLLEGE
WALLAH

Length method

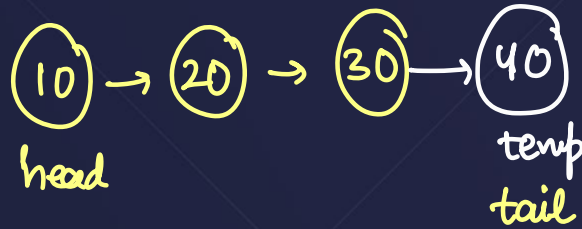
Implement a method to find out the length of a Linked List (Iterative and Recursive)

COLLEGE
WALLAH

function InsertAtEnd ~~method~~ T.C. = $O(1)$

Implement a method to insert a node at the end of a linked list.

Case-1: Size > 0



ll.insertAtEnd(40);

Node* temp = new Node(40);

tail->next = temp

Case-2: Size = 0



ll.insertAtEnd(10)

Node* temp = new Node(10); size++;

head = tail = temp

size++;

tail = temp;

function

InsertAtBeginning ~~method~~

Implement a method to insert a node at the start of a linked list.

Case-1: $Size > 0$



ll. insert At Head (50);

Node* temp = new Node(50);

temp → next = head;

head = temp;

size ++;

Case-2: $Size = 0$

NULL

head

tail

Node* temp = new Node(50);

head = tail = temp;

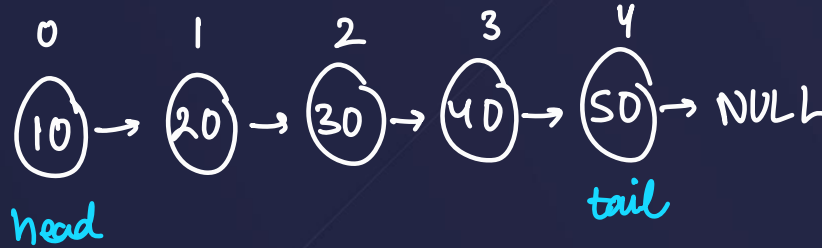
size ++;

Insert method

insertAt(int idx, int val);

T.C. = $O(n)$

Implement a method to insert a node at any given index.



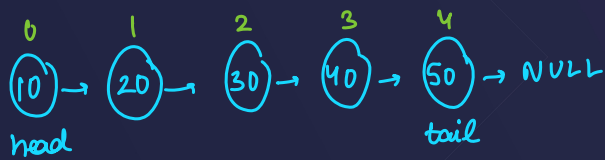
size = 5

if (idx == 0) → ll.insertAtHead

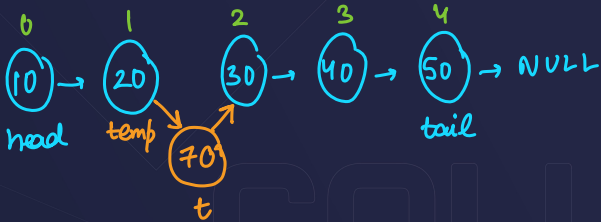
if (idx == size) → ll.insertAtTail

if (idx < 0 || idx > size) → invalid

else



ll.insert At (2, 70);



1) Traverse temp
to idx - 1

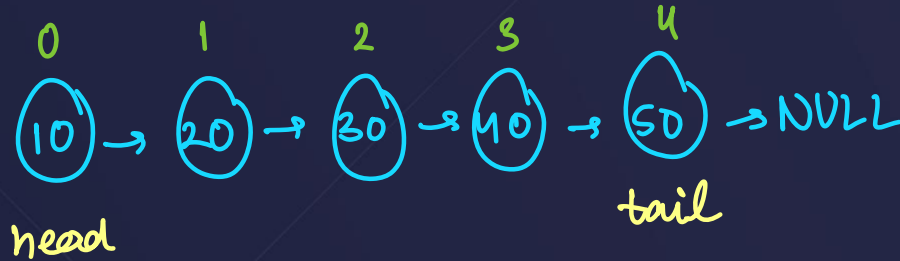
2) $t \rightarrow \text{next} = \text{temp} \rightarrow \text{next}$

3) $\text{temp} \rightarrow \text{next} = t$

getElement method

T.C. = $O(n)$

Implement a method to return the element at any given index of the linked list.

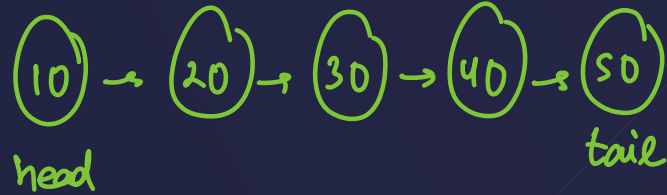


if ($idx < 0$ || $idx \geq size$) invalid else

if ($idx == 0$) return head \rightarrow val

if ($idx == size - 1$) return tail \rightarrow val

An evident limitation of Linked List



```
cout << arr[4];
```

```
arr[4] = 8;
```

COLLEGE
WALLAH

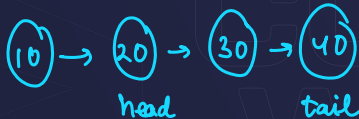
delete At Head :

if (size == 0) → List is Empty

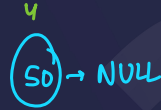
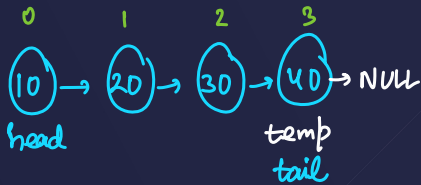
if (size == 1) head = tail = NULL;
size-- NOT REQD.

if (size > 1)

head = head.next;
size--;



delete At Tail :

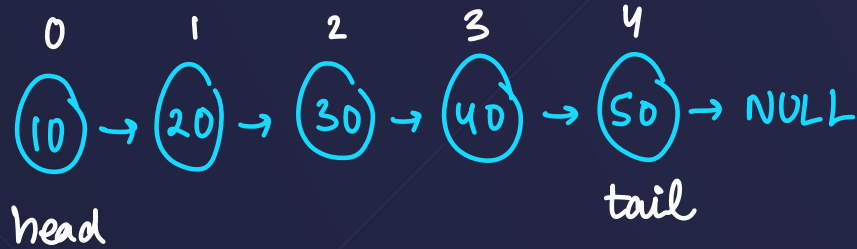


Steps:

- 1) traverse temp such that
temp → next = tail
- 2) temp → next = NULL
- 3) tail = temp

deleteAtIndex method

Implement a function to delete a node at a given index



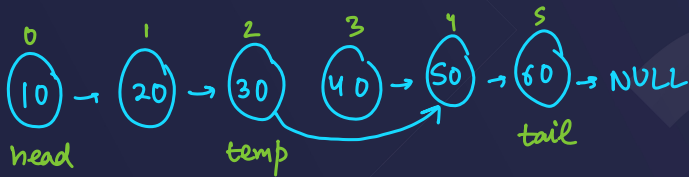
if (idx < 0 || idx >= size) → invalid index

e if (idx == 0) deleteAtHead();

e if (idx == size-1) deleteAtTail();

else {

}



delete AtIdx(3);

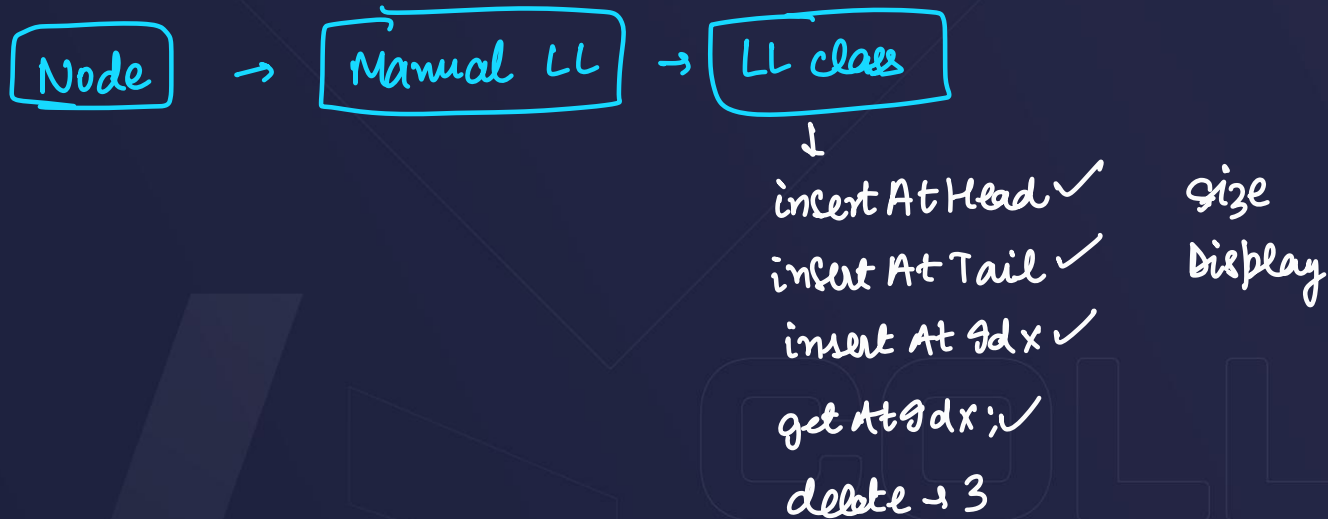
Steps:

1) traverse temp at idx-1

2) temp → next = temp → next → next

Next Lecture

More interesting things about **Linked List**



Homework: Implement LL class but, with only head & size

Next Lecture

More interesting things about **Linked List**

Dher saara

Leetcode



LL → pattern wise

COLLEGE
WALLAH