Lecture :- Linked List: Intro and problems

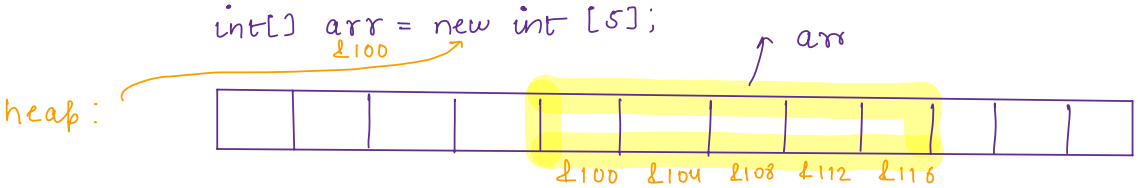
Agenda

- Basics
- Access and search
- Insertion and deletion
- Reverse the LL
- Palindrome.

Introduction | Basics

```
int[] arr = new int [5];
```

&100

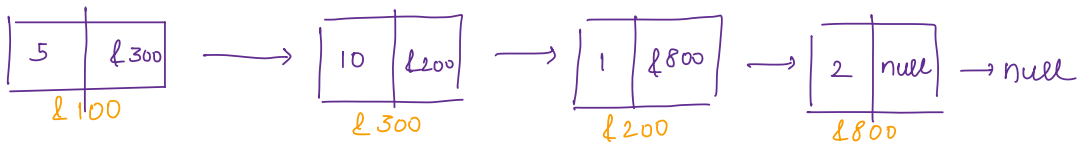



```
int[] arr1 = new int [7];
```
— not possible

[continuous memory allocation]

Linked List — Non-continuous memory allocation

Structure of LL



Eg:- `Node a = new Node(5);`

&100

5	null

```
Node b = new Node(10);
```

&300

10	null

```
Node c = new Node(1);
```

&200

1	null

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

```
a.next = c
```

&100

&200

5	~~null~~
data	next

[5 → 1]

Qu1 find kth element of LL.

$$\overset{0}{2} \longrightarrow \overset{1}{5} \longrightarrow \overset{2}{10} \longrightarrow \overset{3}{3} \longrightarrow \overset{4}{18} \longrightarrow null$$

K=0, ans = 2

k = 3 ans = 3

k=5, ans = null.

$$\overset{0}{2} \longrightarrow \overset{1}{5} \longrightarrow \overset{2}{10} \longrightarrow \overset{3}{3} \longrightarrow \overset{4}{18} \longrightarrow null$$

head

k = 2

```
int getKthElement (Node head, int k) {
    Node hc = head;

    for (i=1; i<=k; i++) {
        if (hc == null) {
            return null;
        }
        hc = hc.next;
    }
    return hc==null ? null : hc.data;
}
```

k=0, ans = 2

K = 4, ans = 18

[B] k = 5, ans = Null pointer exception

[A] k=5, ans = null

k = 6, ans = null

k>=5, ans = null

Dry run: k=5.

hc = head || 2.

2 → 5 → 10 → 3 → 18

i=1, hc = hc.next || 5

i=2, hc = 10

i=3, hc = 3

i=4 hc = 18

i=5, hc = null

Qu2 Given a LL, check if there exists a node with val == x

2 → 5 → 10 → 3 → 1 → null

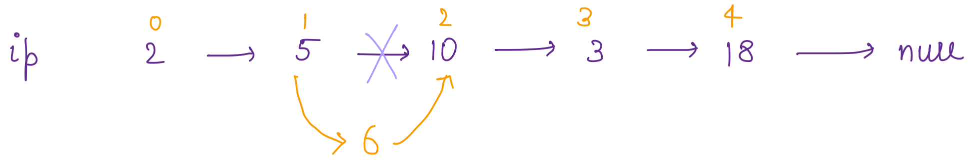
↑
head

x = 10, true

x = 20, false

```
boolean check( Node head, int x){
    Node hc = head;
    while ( hc != null ) {
        if( hc.data == x){
            return true;
        }
        hc = hc.next;
    }
    return false;
}
```

Qu3 Given a LL, insert a node with data = x at k th position and return head after that. [0 <= k <= n]



k=2, data = 6

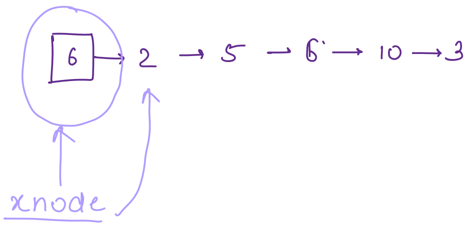
op 2 → 5 → 6 → 10 → 3 → 18 → null

```
Node insert (Node head, int k, int x) {

        Node  xNode = new Node (x);
        if (k==0) {
            xnode.next = head;
            head= xnode;
            return head;
        }
        Node  hc = head;

        for (i=1; i<=k-1; i++) {

            hc= hc.next;
        }

        Node  temp = hc.next;

          hc. next= xNode;

         xNode. next = temp;

         return head;
    }
```

k=0

2 → 5 → 6 → 10 → 3 → 18 → null

k=0, x=6



x.node = head;

head = xnode;

k=5, x = 20

0 1 2 3 4
2 → 5 → 6 → 10 → 15 → null

2 → 5 → 6 → 10 → 15 → 20 → null

[Works]

Qu4 Given a LL, delete the first occurence of node with data = x

2 → 5 → 10 → ~~3~~ → 18 → 3 → 18 → 16 → null

x = 3

2 → 5 → 10 → ~~3~~ → 18 → 3 → 18 → 16 → null

```
hc = head;
while ( hc != null && hc.next != null ) {
    if( hc.next.data == x) {    // hc = 10
        deleteNode = hc.next;
        hc.next = hc.next.next;
        delete( deleteNode);    // C++
        return head;
    }
    hc = hc.next;
}
```

Edgecases: ① if (head == null) {
    return null;
}

② if (head.data == x) {

~~2~~ → 5 → 3 → 2 → null

head

```
    head = head.next;
    return head;
}
```

③ if no x found —— LL will not change

Qu Reverse the LL (pointer wise iteration)

$3 \longrightarrow 18 \longrightarrow 5 \longrightarrow 6 \longrightarrow 2 \longrightarrow 18 \longrightarrow null$

$\downarrow$ reversal

$null \leftarrow 3 \leftarrow 18 \leftarrow 5 \leftarrow 6 \leftarrow 2 \leftarrow 18$

Step1:

null ← 3

prev = null

curr = 3

agla = curr.next || 18

curr.next = prev || 3 → null.

prev = curr || 3

curr = agla || 18.

Step2: null ← 3 ← 18

prev = 3

curr = 18

agla = curr.next || 5

curr.next = prev || null ← 3 ← 18

prev = curr || 18

curr = agla || 5

Step3: null ← 3 ← 18 ← 5

prev = 18

curr = 5

agla = curr.next;

curr.next = prev; null ← 3 ← 18 ← 5

prev = curr;

curr = agla;

⋮ ⋮

stop when curr = = null.

```
Node reverse( Node head) {
```

head of reverse LL

```
    prev = null;
    curr = head;
    while( curr != null) {
        agla = curr.next;
        curr.next = prev;
        prev = curr;
        curr = agla;
    }
    head = prev;
    return head;
}
```

Qu check if LL is a paindrome. [cannot create diff LL]

Eg: $1 \to 3 \to 5 \to 1 \to 3$, No

$1 \to 3 \to 5 \to 3 \to 1$, yes

arr: [1 3 5 3 1]

i i i j j

right → · next

j-- ← left

Eg: 1.)

$\underbrace{\overset{0}{1} \to \overset{1}{2} \to \overset{2}{4} \to \overset{3}{5}}_{\text{Part1}} \to \underbrace{\overset{4}{5} \to \overset{5}{4} \to \overset{6}{2} \to \overset{7}{1}}_{\text{Part2.}}$

↓ rev

$1 \to 2 \to 4 \to 5$ (i)

$\underbrace{1 \to 2 \to 4 \to 5}_{\text{rev(Part2)}}$ (j)

2.)

$\underbrace{\overset{0}{1} \to \overset{1}{2}}_{\text{Part1}} \to \underbrace{\overset{2}{3} \to \overset{3}{2} \to \overset{4}{1}}_{\text{Part2}}$

↓ rev (part2)

$1 \to 2 \to 3$ (i i)

$1 \to 2$ (j j)

Step 1: Length of LL.

```
int len( Node head) {
    int cnt = 0;
    Node hc = head;
    while (hc != null) {
        cnt ++;
        hc = hc.next;
    }
    return cnt;
}
```

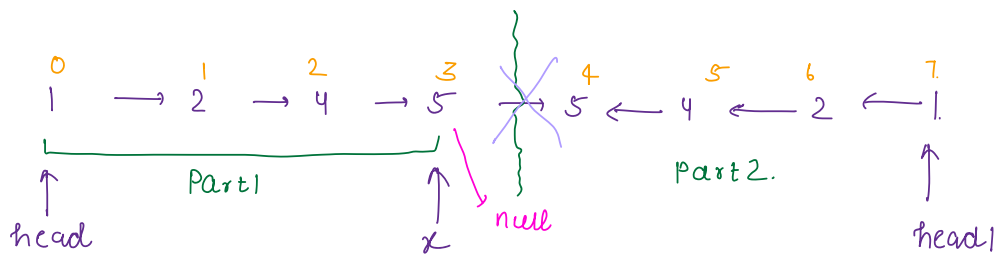
2.) Travel till $\frac{len-1}{2}$ and say node = x.

$x = getKthElement\left(head, \frac{len-1}{2}\right);$

3.) Head of part 2 :- x.next

head1 = x.next; x.next = null

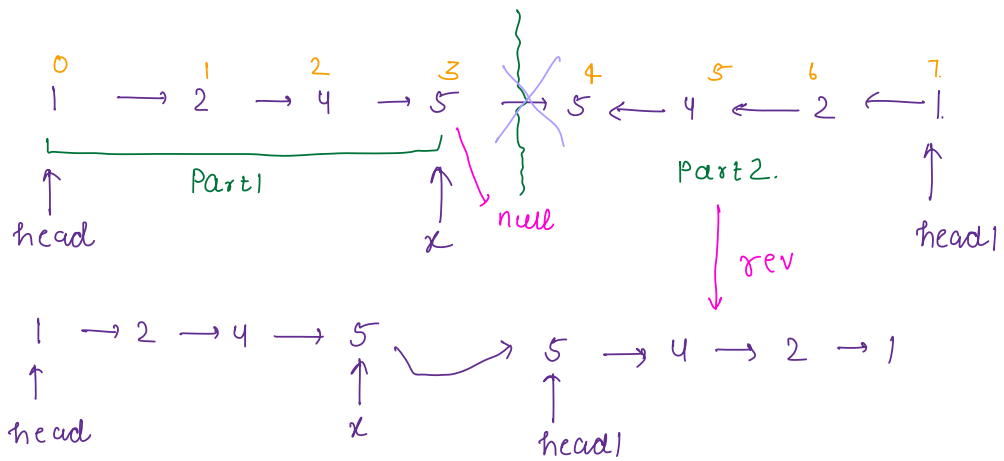
reverse (head1);



Part1 →	1	2	4	5	null
Part2 →	1	2	4	5	null

```
Node i = head;
Node j = head1;
while ( i != null && j != null) {
    if ( i.data != j.data) {
        return false;
    }
    i = i.next;
    j = j.next;
}
return true;
```

4. Restore the LL




```
head1 = reverse (head1);
x.next = head1;
```

Thankyou ☺