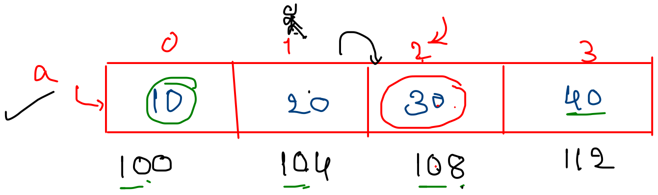
S4-Class2 [Linked List-1]

System Defined Data Types :-

inbuilt DT / primitive DT

int, float, String, etc ....

let
Base address = 100, size(int) = 4B



⇒ size(a) = 16B

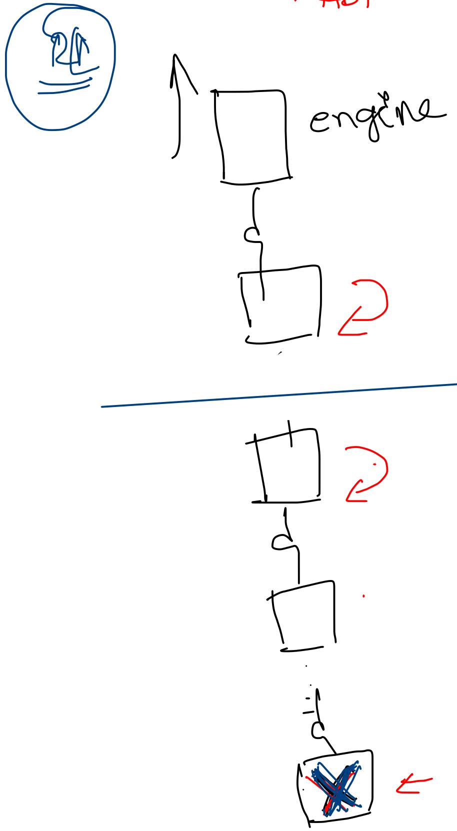
$P(a[2]) \rightarrow 30$

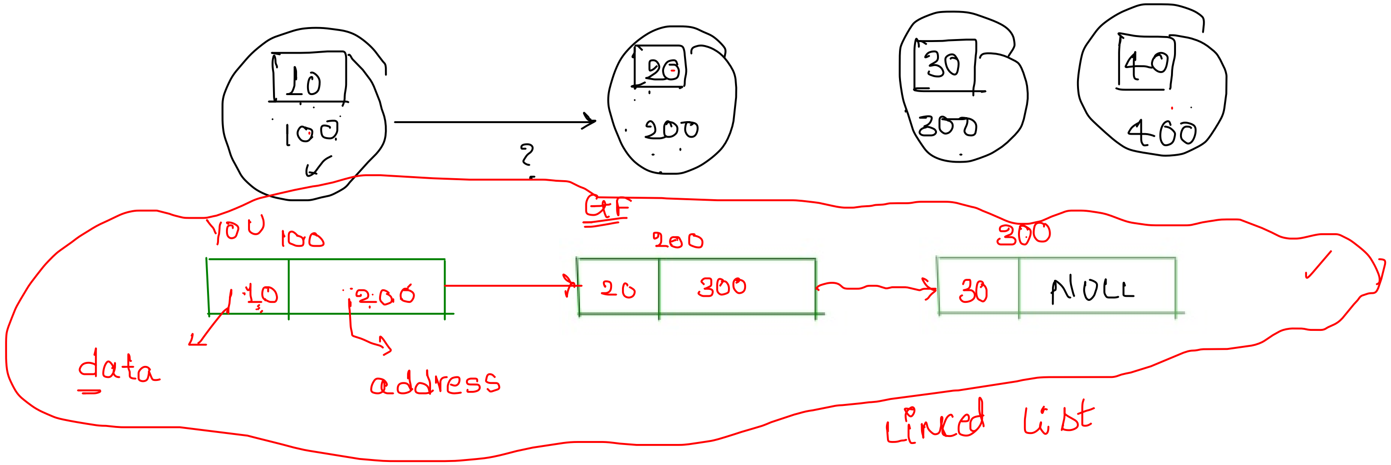
$100 + (2-0) * 4 = 100 + 8 = 108$

Random Access
is possible } O(1)

User Definied Data Types :-

↳ ADT





Node obj1 = new Node();

Node obj2 = new Node();

AIOBE ✓

SOFE ✓

NOSUCHELEMENT

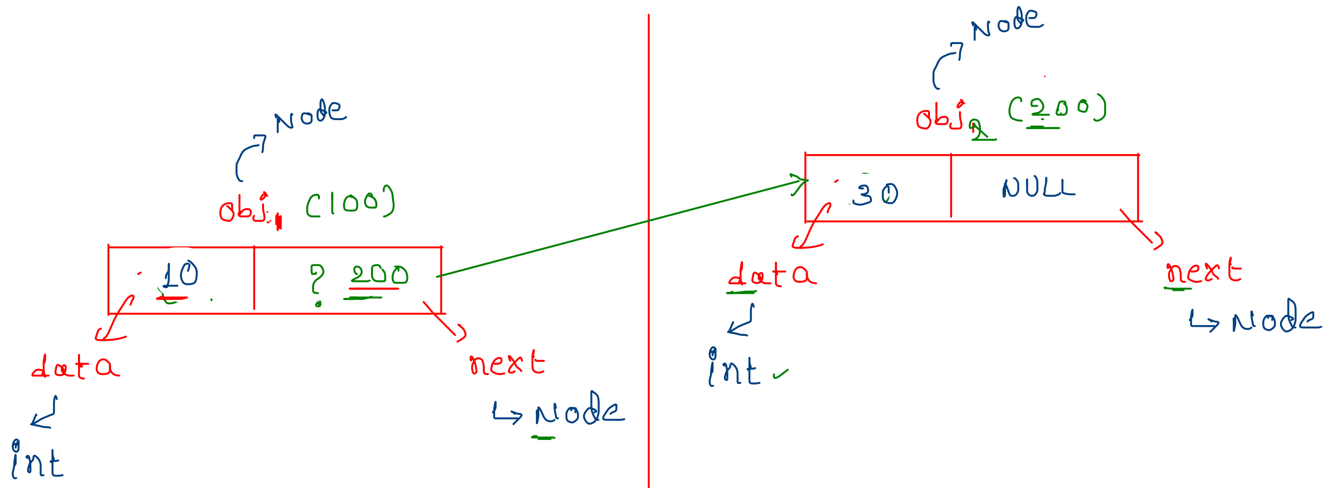
NOSUCHCLASS

InputMissMatch,

~~Node head;~~

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

elements of a class



print(obj1.data); → 10

print(obj1.next); → 200

print(obj1); → 100

print(obj2.data); → 30

print(obj2.next); → NULL ✓

null. something

print(obj2); → 200

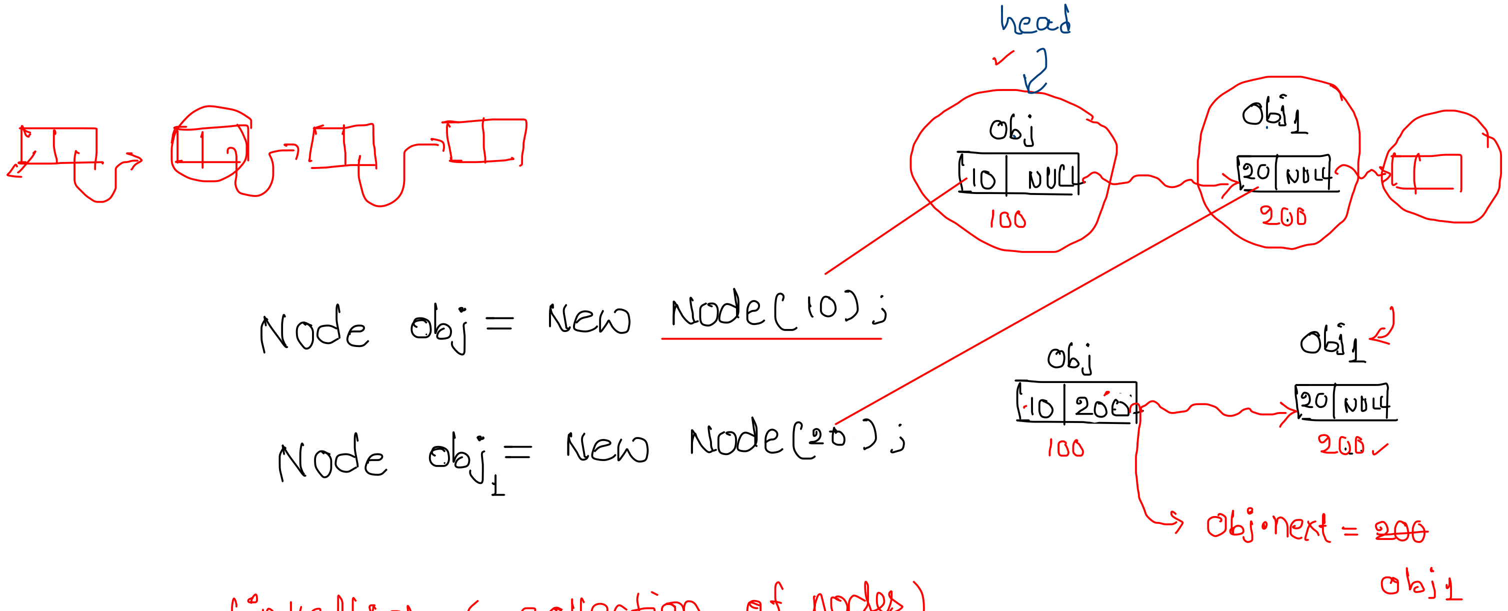
print(obj2.next . data)

NULL.

* NULL pointer Exception

LinkedList obj=new LinkedList();

```
class LinkedList
{
        Node head;
        class Node
        {
                ~~int d~~ata;
                Node next;
        }
        Node(int d)
        {
                data=d;
        }
}
```

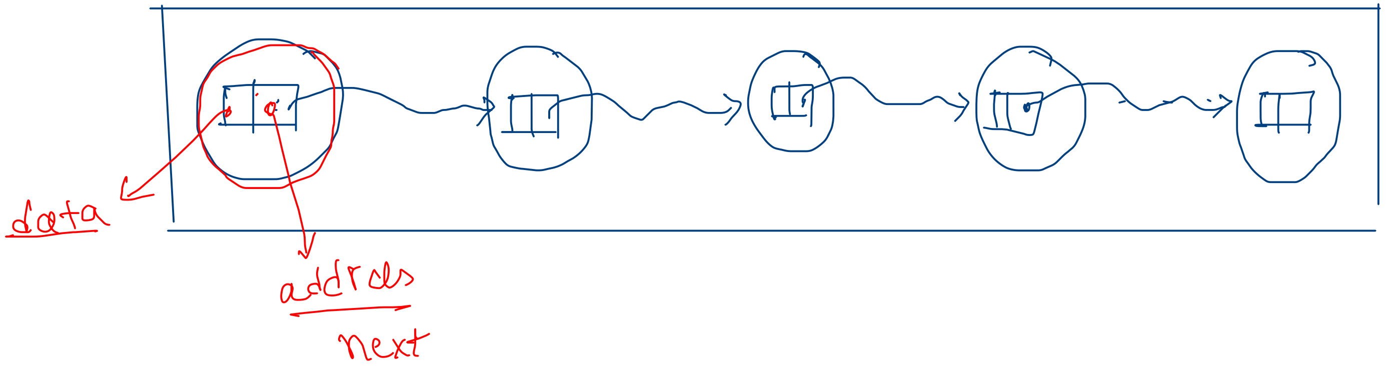


Node obj = New Node(10);

Node obj_1 = New Node(20);

obj.next = ~~200~~ obj1

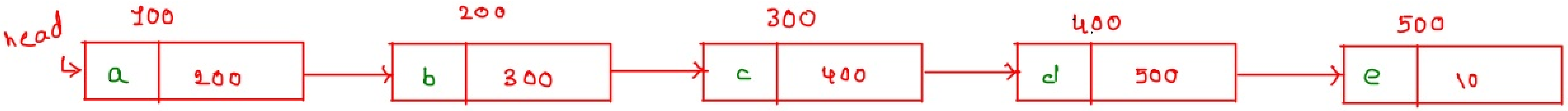
LinkedList (collection of nodes)

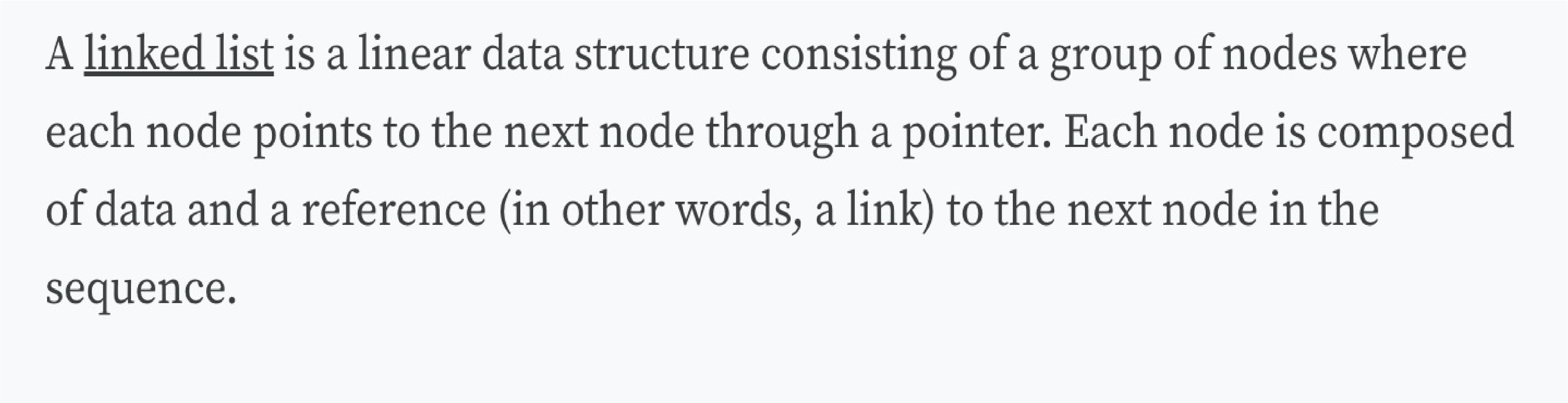


<u>LL</u> (SLL)

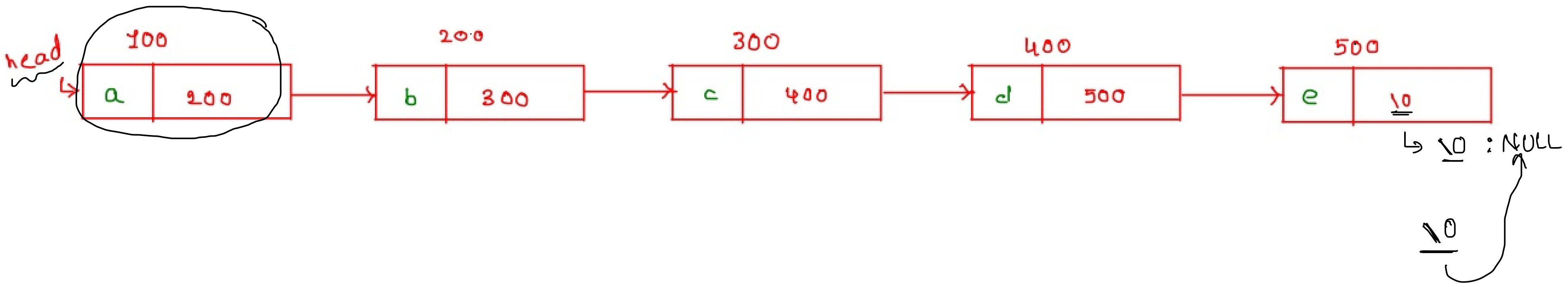
$\underbrace{100, 200, \cdots 500}_{\text{addresses}}$ (Type : Node)

$\underbrace{a, b, \cdots e}_{\text{data}}$ (Type : char)





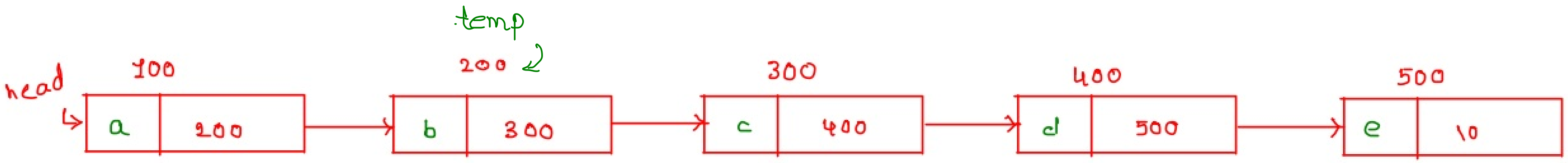
A linked list is a linear data structure consisting of a group of nodes where each node points to the next node through a pointer. Each node is composed of data and a reference (in other words, a link) to the next node in the sequence.



Note:-

-> Head pointer will always points to begnning of the list

-> Last node address part is always NULL in a Single linked list



1) print(temp) → 200

2) print(temp.data) → b

3) print(temp.next) → 300

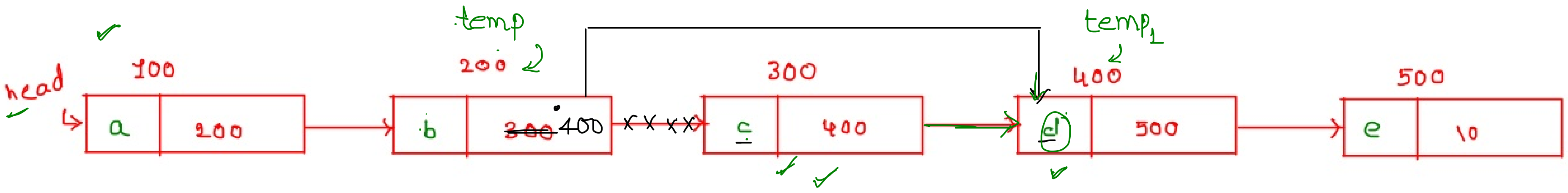
200 ↳

print(temp.next.next.data) →

200 300 40 ↳ d ✓

print(temp.next.data) →

200 300 ↳ c ✓



temp.next = temp1 ✓

print(temp.next.next.data) ✓

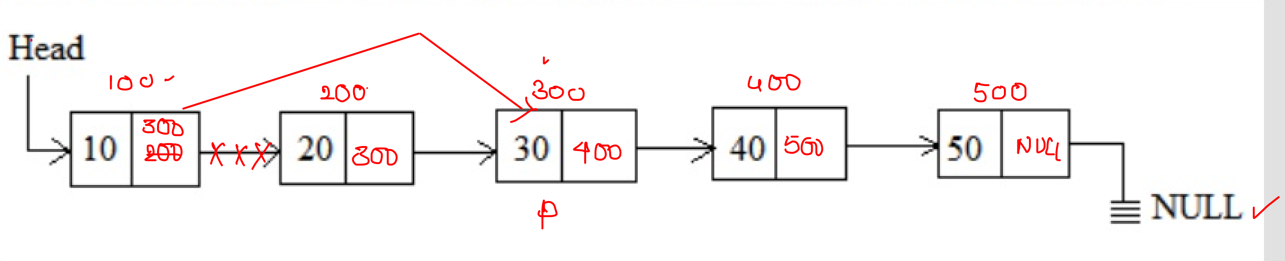
200 400 500 ↳ e ✓

Linked List :- Please take care of Null Pointer Exception

null. something ==> if you try to access then it gives NPE

null.data or null.next

1. Given a linked list L with head pointing to the first node of L, shown below:



What is the output when the following sequence of operations applied on the given linked list? ✓

P is a node pointer

(i) P = $\underset{100}{\underline{\text{head}}} \rightarrow \underset{200}{\underline{\text{next}}} \rightarrow \underset{300}{\underline{\text{next}}}$; ✓

(ii) head → next = P;

(iii) printf("%d", $\underset{100}{\underline{\text{head}}} \rightarrow \underset{300}{\underline{\text{next}}} \rightarrow \underset{400}{\underline{\text{next}}} \rightarrow \underline{\text{data}}$); ↳ 40 ✓

The output of the following code is ______________ (Marks: 0.00)

Let's see few operations to understand Linked List better

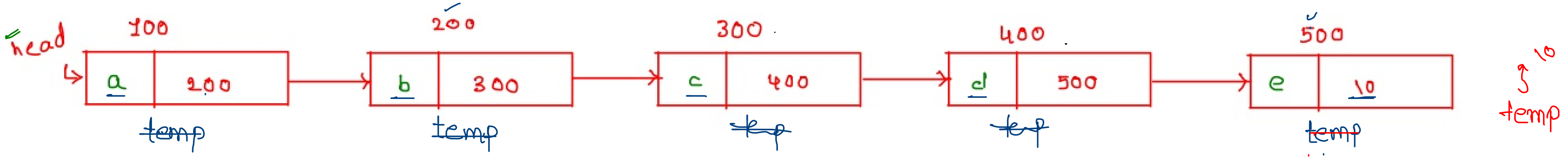
Focus on how linking is happening [NOT on CODE, PLEASE ...]

Traversing a Linked List / print all elements of list

All probs: OJ
LL

→ fn. complete
(i/p reading x)

Unit-1

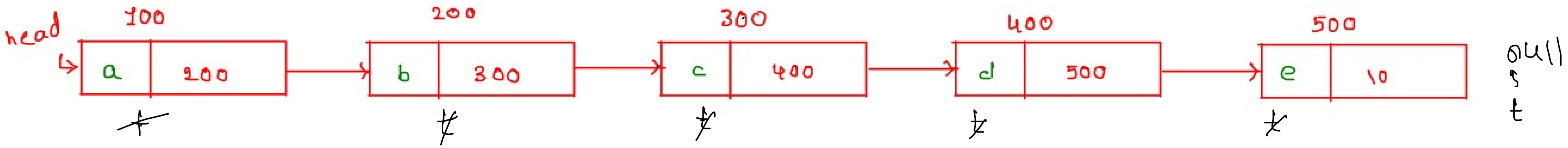


↳ p(temp) → 500
p(temp.data) → e
p(temp.next) → null

o/p a, b, c, d, e

```
void printList(Node head)
{
        Node temp=head;
        while(temp!=NULL)
        {
                System.out.println(temp.data);
                temp=temp.next;
        }
}
```

```
int findLength(Node head)
{
        int count=0;
        Node temp=head;
        while(temp!=null)
        {
                count++;
                temp=temp.next;
        }
        return count;
}
```



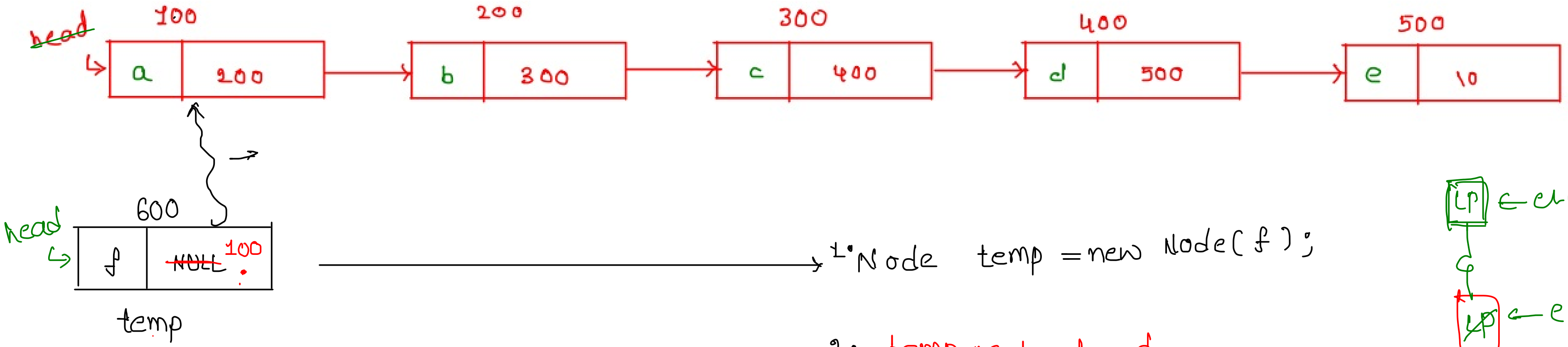
c = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

```
int findLength(Node head) ✓
{
        int count=0;
        Node temp=head;
        while(temp!=null)
        {
                count++;
                temp=temp.next;
        }
        return count;
}
```

o/p :- f → a → b → c → d → e ✓

1) Adding an Element at the begnning

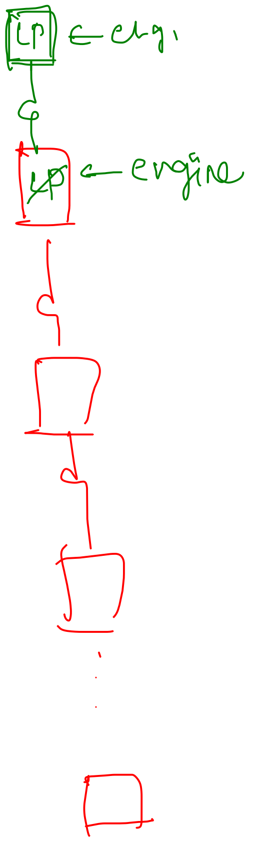
f



1. Node temp = new Node(f);

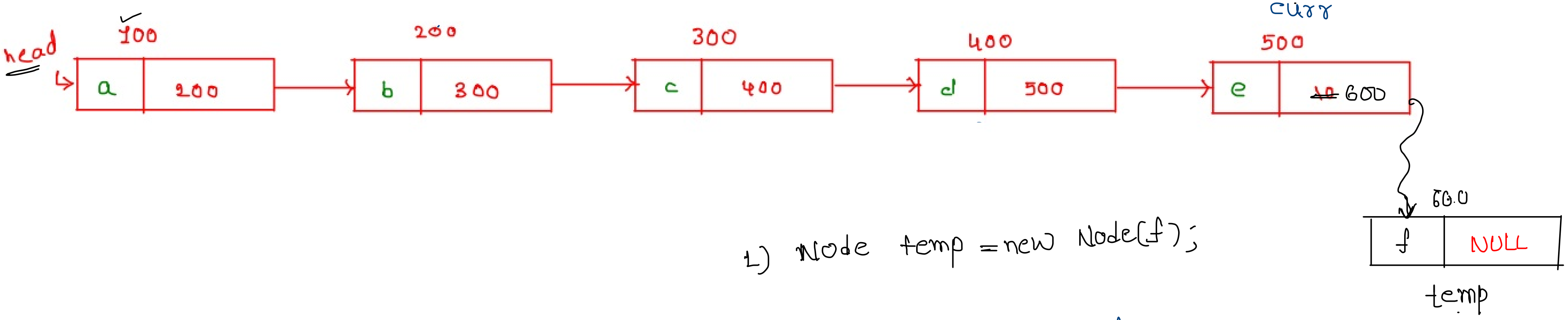
2. temp.next = head

3. head = temp



2) Adding an Element at the end

(f)



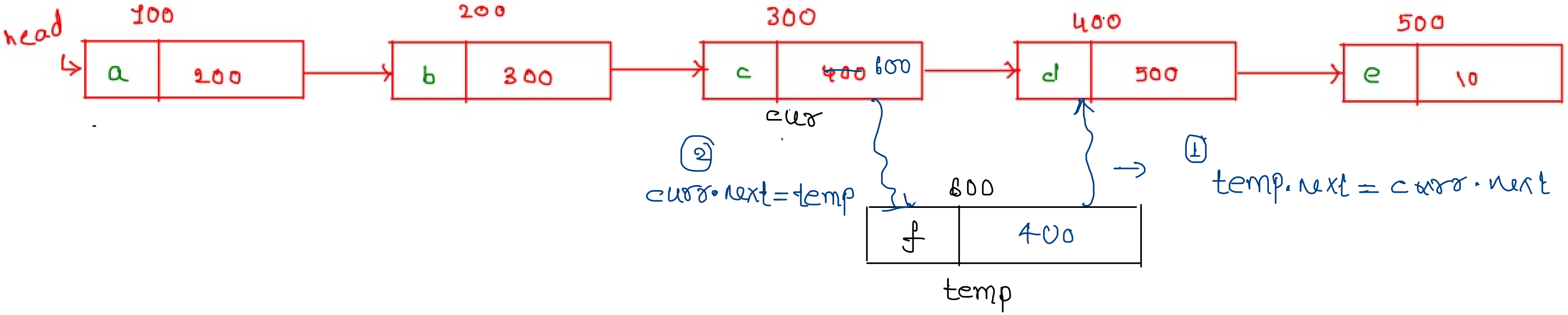
```
1) Node temp = new Node(f);

2) Node curr = head;

   while( curr.next ! = NULL)  x
   {
        curr = curr.next

   }

3) curr.next = temp

4) return head.
```

O/p :- a → b → c → f → d → e

3) Adding an Element after a particular element

(f)

c → Always present



X { 1) curr.next = temp ✓

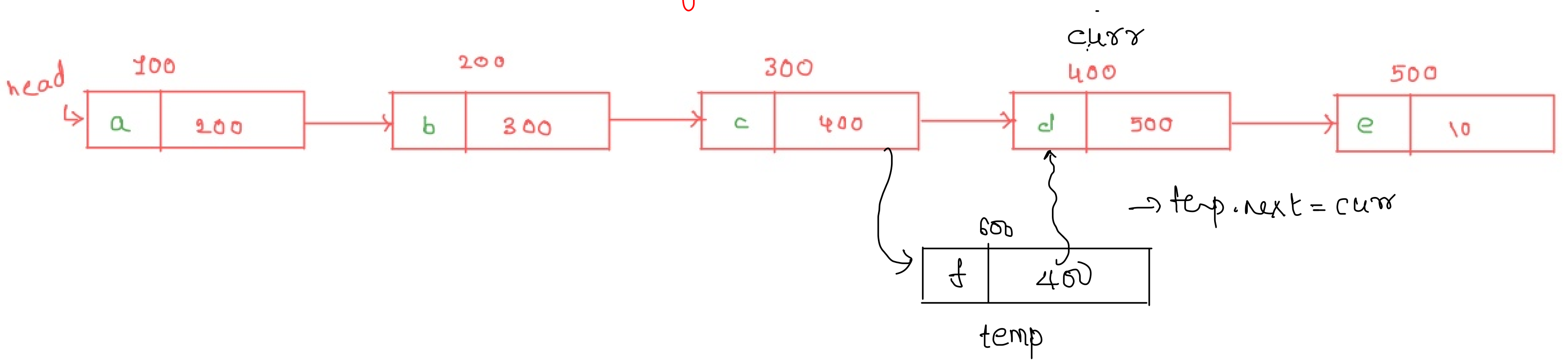
2) temp.next = curr.next }

a → b → c → f → d → e

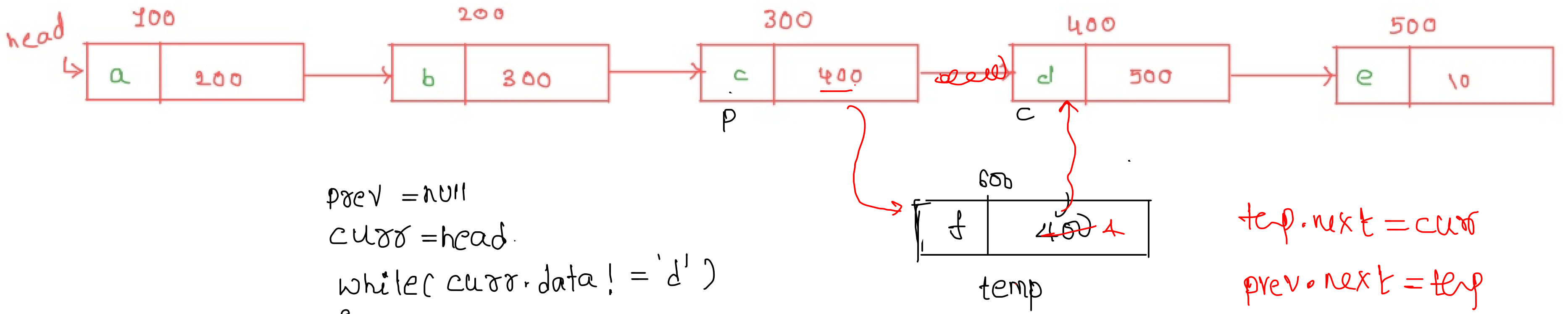
4) Adding an Element before a particular element

(f)

d ↳ Always present.



→ temp.next = curr



```
prev = null
curr = head
while( curr.data ! = 'd' )
{
        1. prev = curr;
        2. curr = curr.next;
}
```

temp.next = curr

prev.next = temp