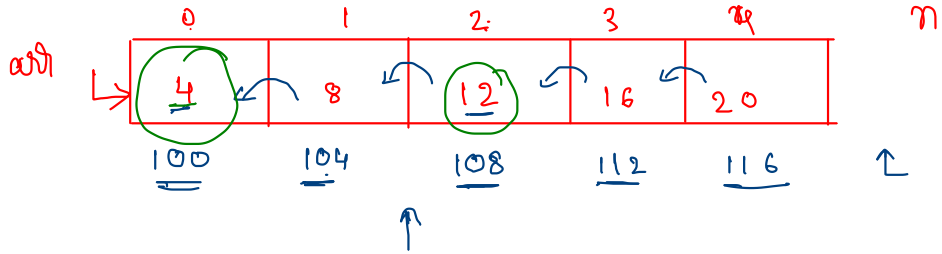
Linked List

Arrays → integer's



n = 5

if (int) → 4B

1 ele = 4B.

2 ele = 8B

10 ele = 40B.

} Array's.

Base — Address.

print(arr[2]) ⇒ 12

Size(int) = 4B.

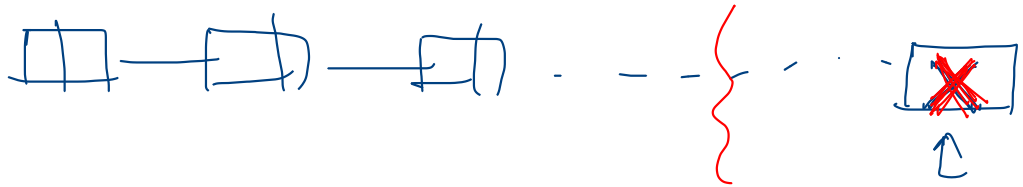
↳ 100 + (2-0) * 4 (size) = 100 + 8

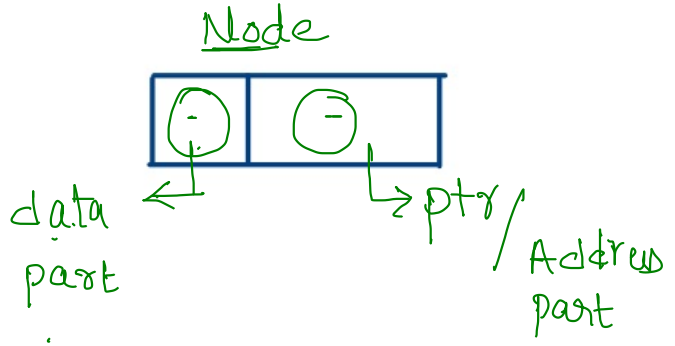
BA

= 108

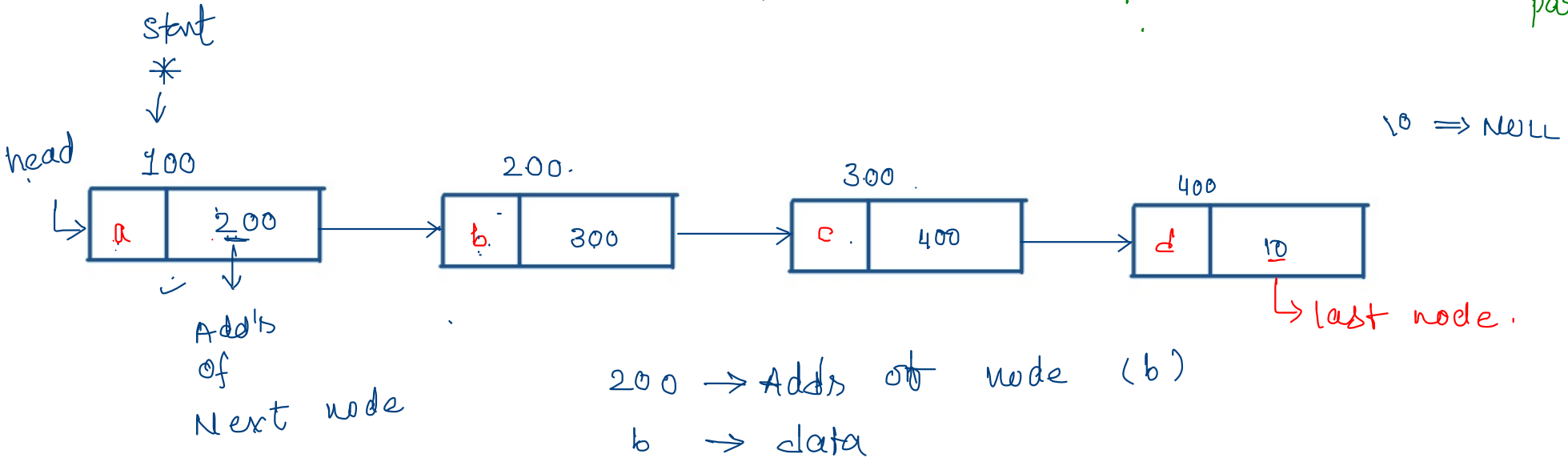
↓

# of elements to cross



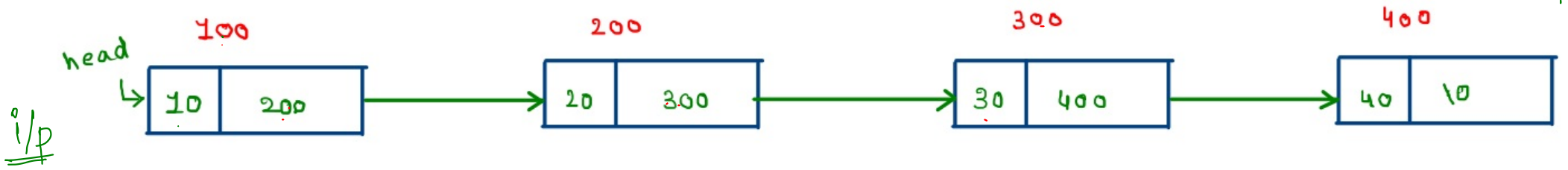


•next ⇒ add's of node



If any node address is null, which means it is the last node of that linked list

Let's assume 100, 200, 300, 400 are addresses of nodes 10, 20, 30 and 40 respectively



i/p

① print(head) ⇒ 100 ✓

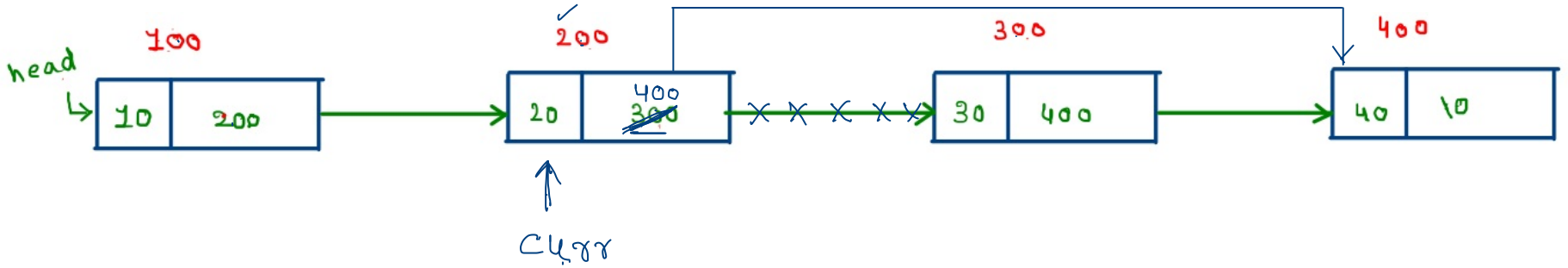
② print(head.data) ⇒ 10
100

③ print(head.next.next.data)
100 200 300 ↳ 30

* Access the data; use .data
Port of node

data ⇒ .data

Add's ⇒ .next



1) Node curr=head.next ✓

100 200

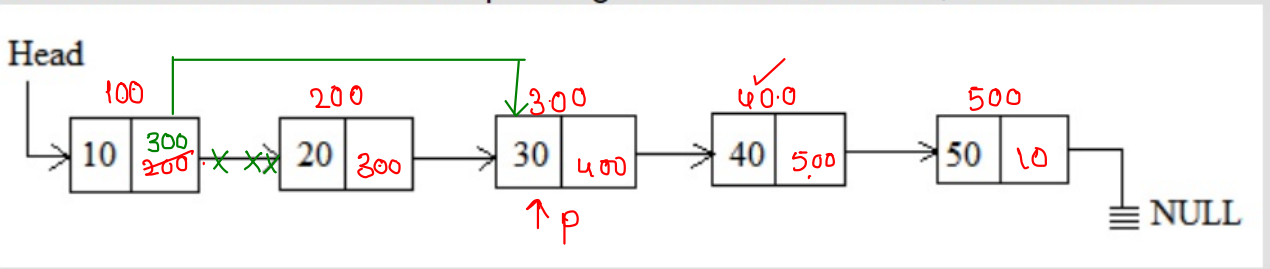
→ 2) curr.next=head.next.next.next ✓

100 200 300 400

3) print(curr.next.data) ⇒ o/p

200 400 → 40

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

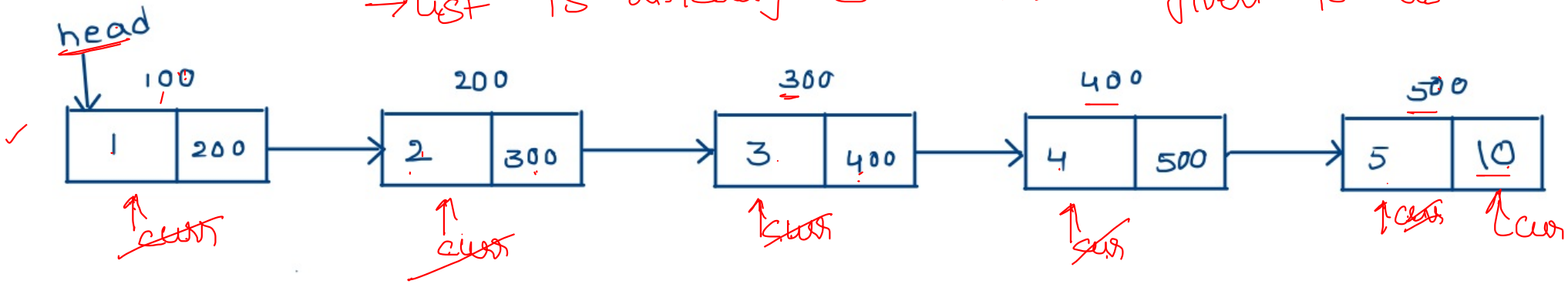
(i) P = head → next → next; ⇒ p = 300

(ii) head → next = P;

(iii) printf("%d", head → next → next→ data);

The output of the following code is 40 (Marks: 0.00)

→ List is already created ✓ given to us.



—

O/P

1, 2, 3, 4, 5

```
printList(Node head)
{                          100
   Node curr=head
  if(curr==null) ⇒ empty list
       return ✓
  while(curr!=null)
   {
     console.log(curr.data) ✓
     curr=curr.next ✓
   }          100   200
}
```

1, 2, 3, 4, 5,

⇒ Array Elements.

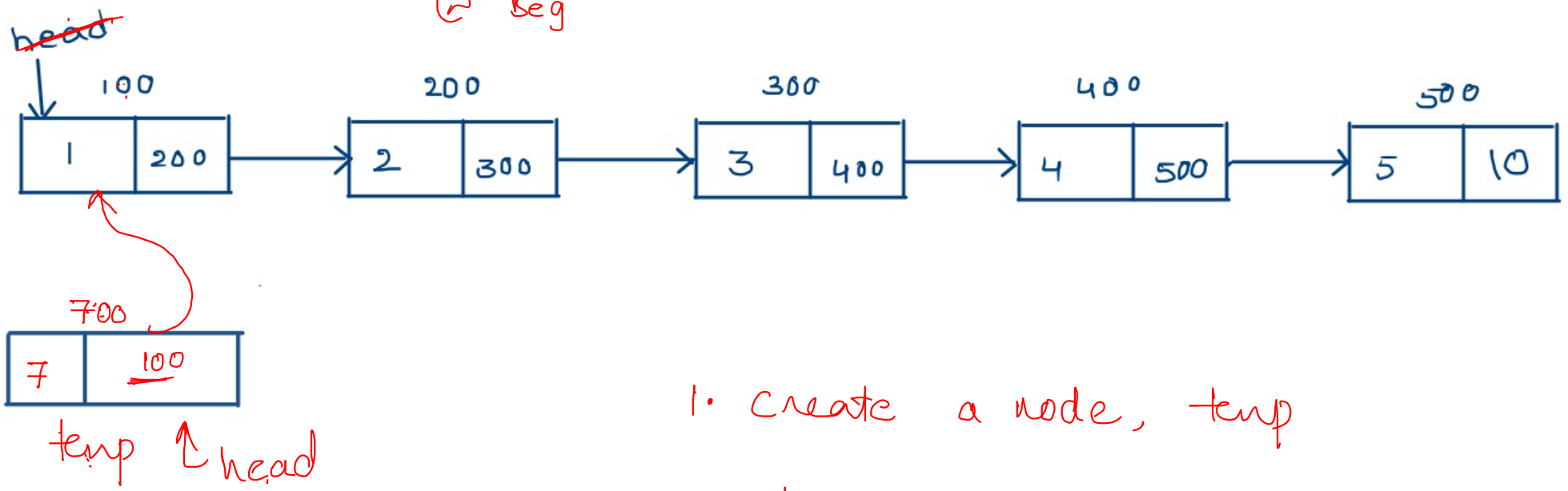
```
for(i=0; i<n; i++)
{
     print( arr[i])
}
```

2) Insert At the Beg of SLL (7)

o/p

⇒ 7, 1, 2, 3, 4, 5

② Beg

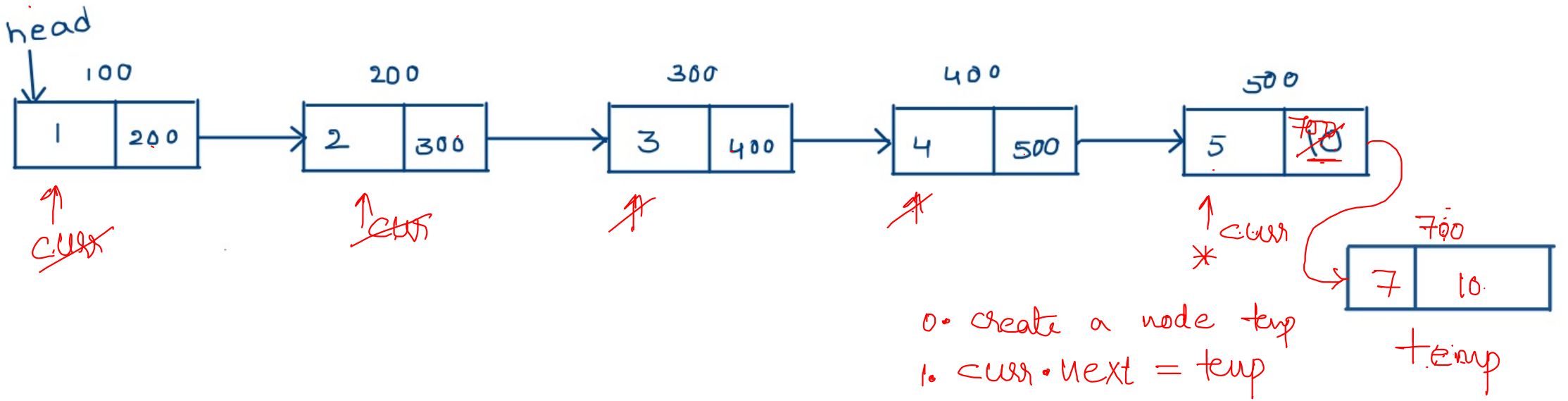


1. create a node, temp
2. temp.next = head.
3. head = temp ✓

```
Node insertAtBeg(head,data)
{
    1.create a node with the name curr
    2.curr.next=head
    3.head=curr
}
```

o/p

3) Insert a node at the end of SLL (7, @ end) 1, 2, 3, 4, 5, 7



0. create a node temp
1. curr.next = temp
2. temp.next = 10

```
Node insertEnd(head,data)
{
    1.create a node with the name temp
    2.Node curr=head
    3. while(curr.next!=null)
            curr=curr.next
    4. curr.next=temp
    5. temp.next=\0

}
```

4) Insert after particular element

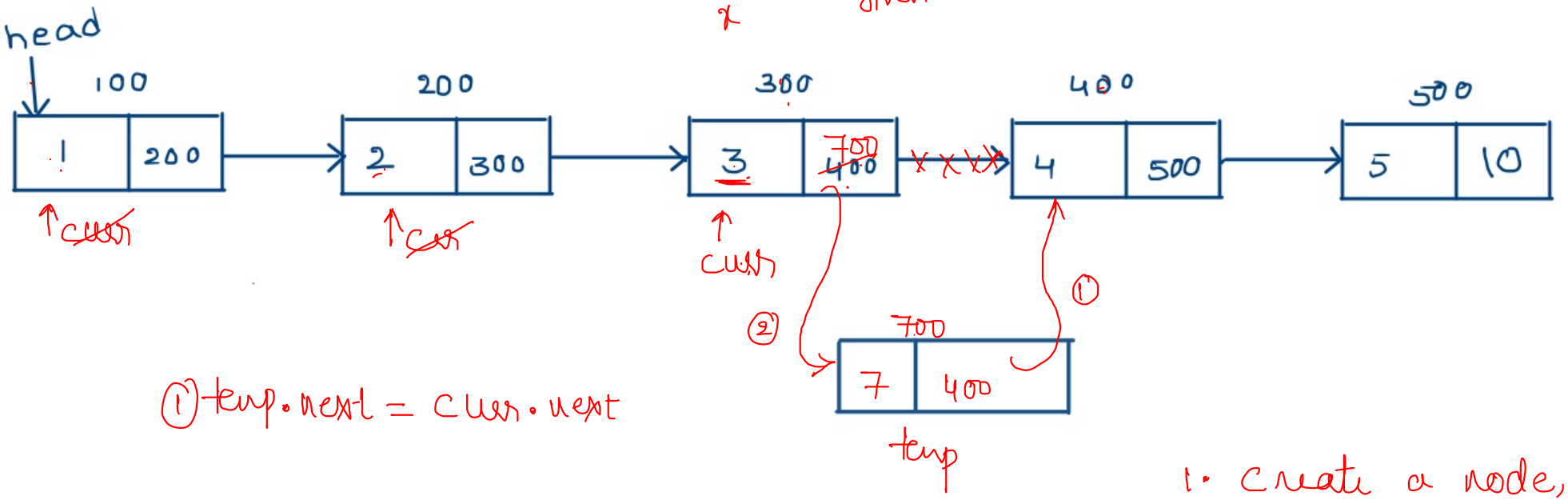
given (3), (7) given

insert

o/p

1, 2, 3, 7, 4, 5

⇒



① temp.next = curr.next

② curr.next = temp

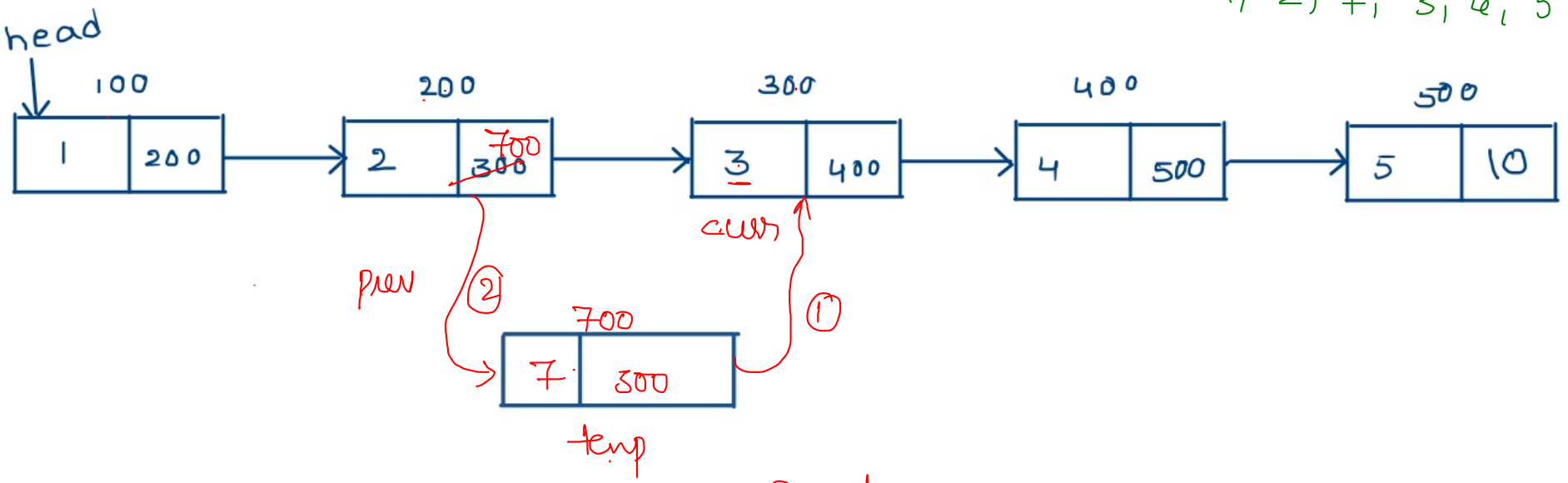
1. create a node, temp

Node insertAfter(head,element,data)
{
1.Node curr=head
2. move curr pointer to the node after which you want to add ⇒ by using loop
3. create a node with the name temp
4. temp.next=curr.next
5. curr.next=temp
}

insertBefore Particular element (ele – 3), want to insert (7)

o/p

1, 2, 7, 3, 4, 5



1. temp.next = curr

last step → 2. prev.next = temp

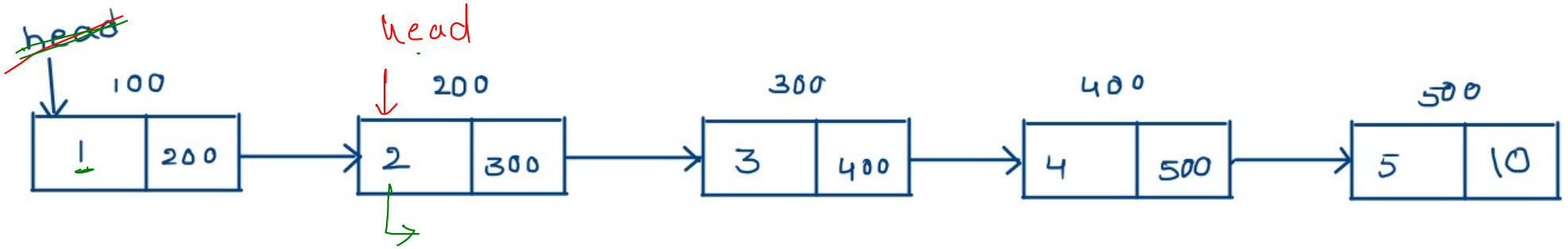
```
Node insertBefore(head,element,data)
{
    if(head==null)
       return
    if(isPresent(head,element)==false)
       return
    Node temp=new Node(data)
    Node curr,prev
    for(curr=head;curr!=null;prev=curr,curr=curr.next)
    {
        if(curr.data==element)
        {
            temp.next=curr
            prev.next=temp
            break;

        }
    }
    return head
}
```

```
function isPresent(Node head, x)
{
        if(head==null)
                return false
        Node curr=head
        while(curr!=null)
        {
                if(curr.data==x)
                        return true
        }
        return false
}
```

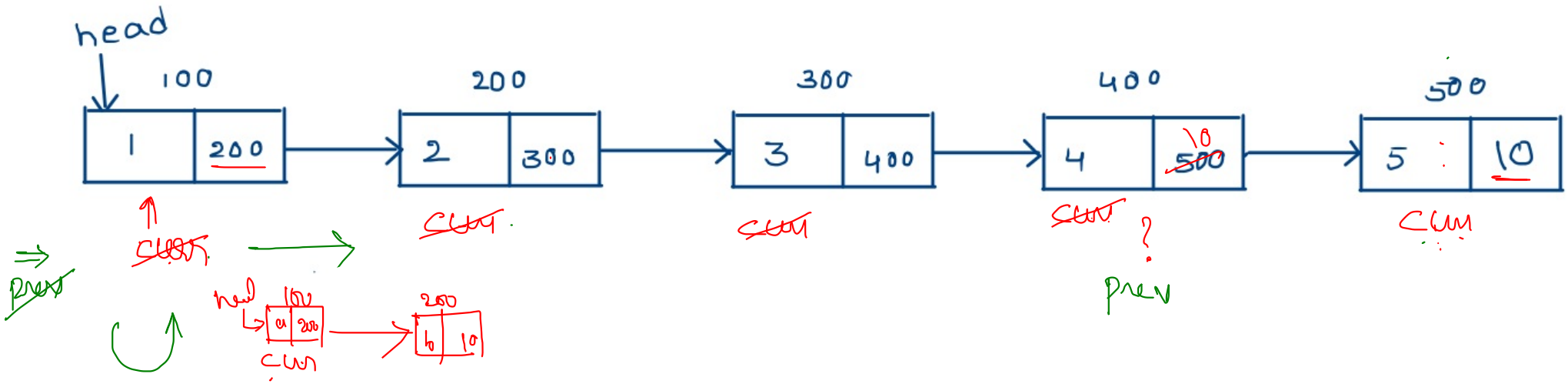
6)Delete at the beg of SLL

⇒op :- 2, 3, 4, 5



7)Delete at the end of SLL

op 1, 2, 3, 4



⇒ (curr.next.next == null)

1. Move curr ptr to the last node (88)
   prev ptr to last but one node.

2. prev.next == 10 ✓ null.

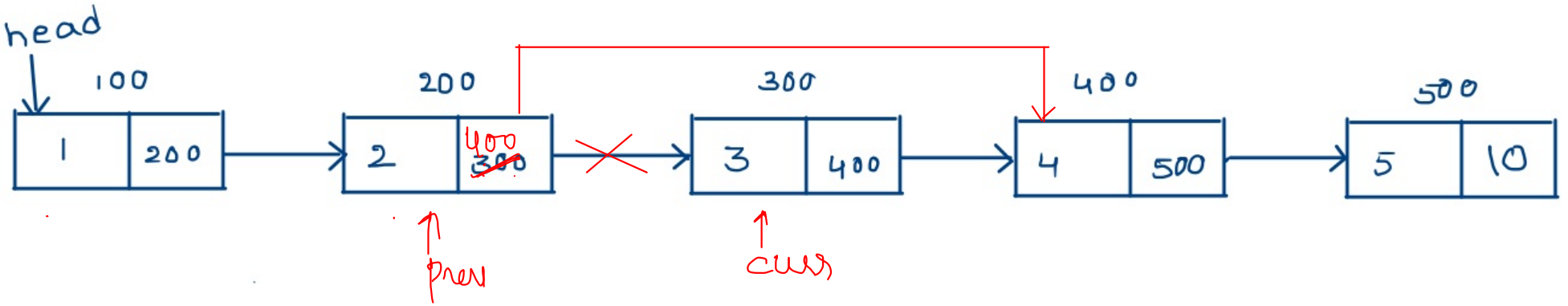
```
prev = null
while(curr.next != null)
{
        prev = curr
        curr = curr.next

}
prev.next = null
```

o/p

1, 2, 4, 5 ✓

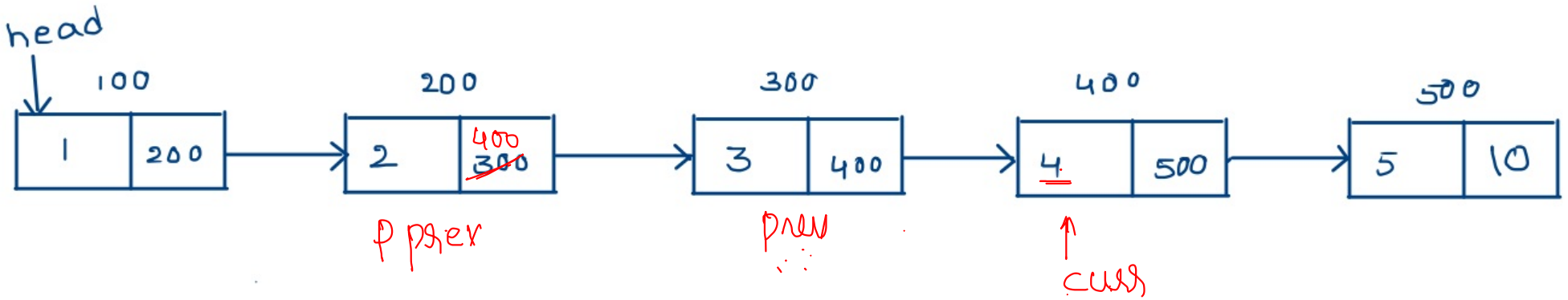
8)Delete a particular node (3)



Prev.next = curr.next ✓

```
//assume that SLL is not empty & element is present
Node deleteElement(head,element)
{
    Node curr=head,prev=null
    for(curr=head;curr!=null; prev=curr, curr=curr.next )
    {
        if(curr.data==element){
            prev.next=curr.next; break;}
    }
    return head
}
```

9)Delete a node before particular element (4) ⇒ 1, 2, 3, 5 ✓



pprev.next = curr

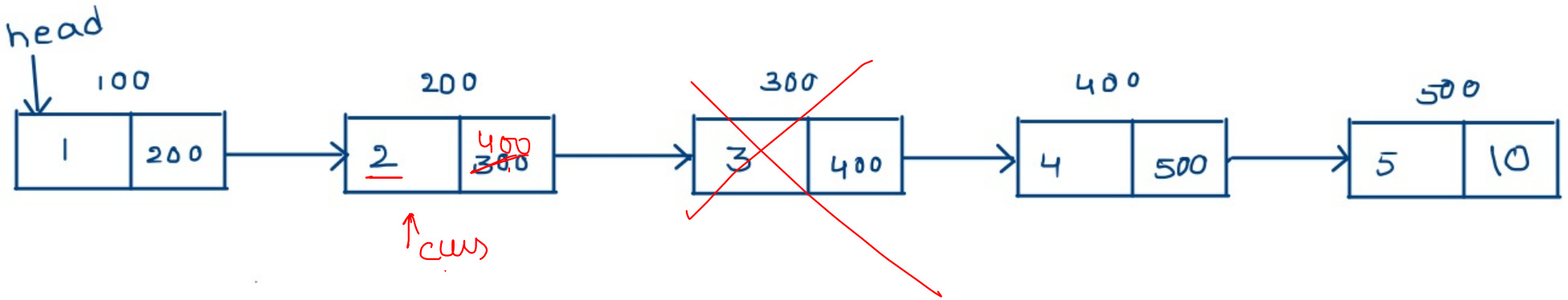
↳ prev.next

```
Node deleteBefore(head,element)
{
    Node curr,prev=null,pprev=null
    for(curr=head;curr!=null;pprev=prev,prev=curr,curr=curr.next)
    {
        if(curr.data==element)
        {
            pprev.next=curr // pprev.next=prev.next
            break;
        }
    }
    return head
}
```

10)Delete a node after particular element (2)

o/p

1, 2, 4, 5 ✓



cur.next = cur.next.next ✓

(END - END) ⇒ 1st year C-prog ✓

Insertion

- → @ Beg.
- → @ end
- → @ before ele.
- → @ after ele

Deletion

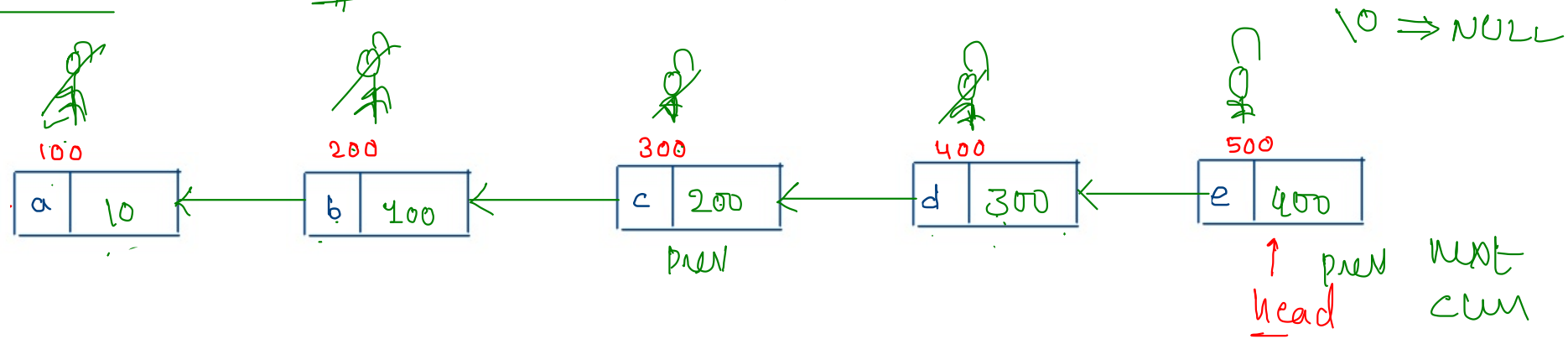
- → @ Beg
- → @ end.
- → @ Particular ele.
- → @ Before ele
- → @ after ele.

I/p :- a → b → c → d → e

O/p :- e → d → c → b → a ✓

* * *

① Reverse the SLL :-



Prev = null, next = null

curr = head.

x → while (curr != null)

{

1. next = curr.next

2. curr.next = prev

3. prev = curr

4. curr = next

}

head = prev; ret. head.

```
function  reverse(node) {
   var  prev = null;
   var  current = node;
   var  next = null;
        while  (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        node = prev;
        return  node;
   }
```

Find the middle Node in the given SLL

```
function printMiddle(Node head)
 {
        Node slow_ptr = head;
        Node fast_ptr = head;


            while  (fast_ptr != null  && fast_ptr.next != null)
            {
                fast_ptr = fast_ptr.next.next;
                slow_ptr = slow_ptr.next;
            }
            print(slow_prt.data)

}
```