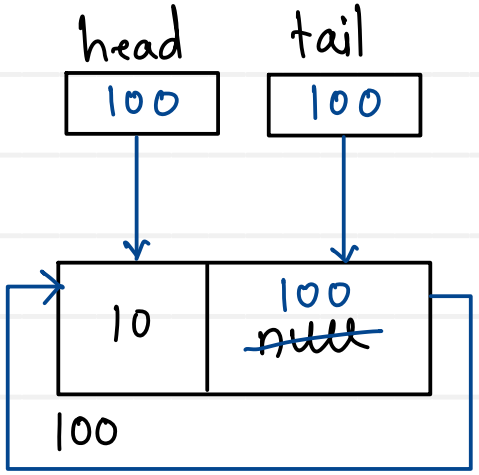
Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

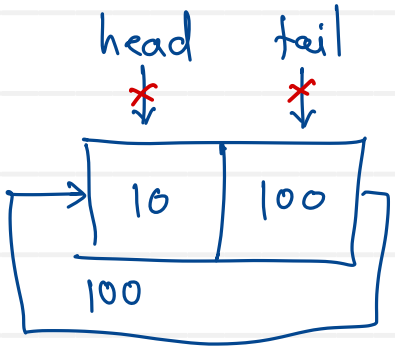
Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

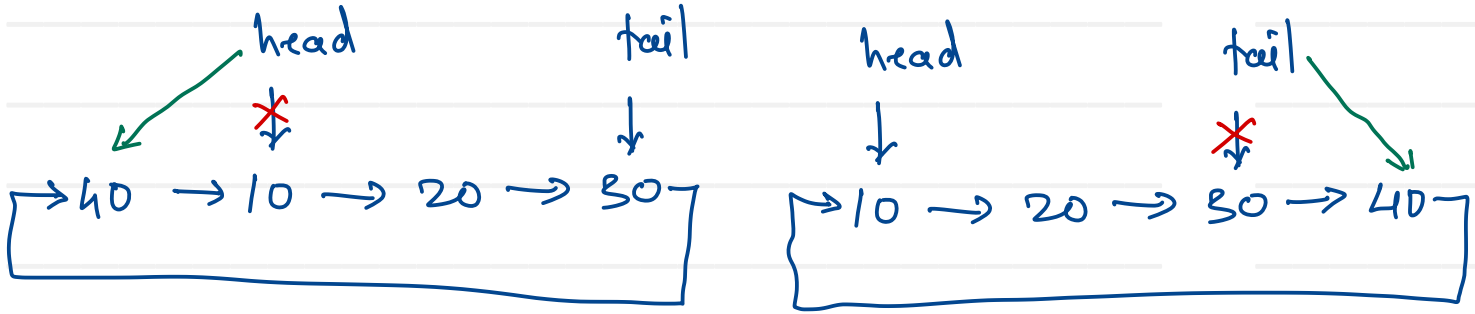



```
if (head == null) {
    head = tail = newnode;
    tail.next = head
}
```

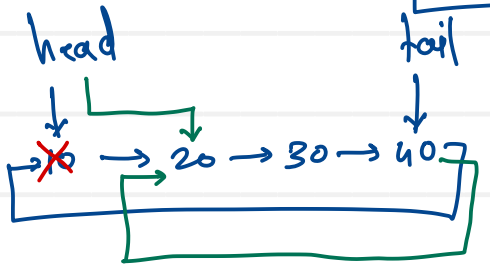

```
if (head == null)
    return;
```




```
if (head == tail)
    head = tail = null
```



Irland of Isolation

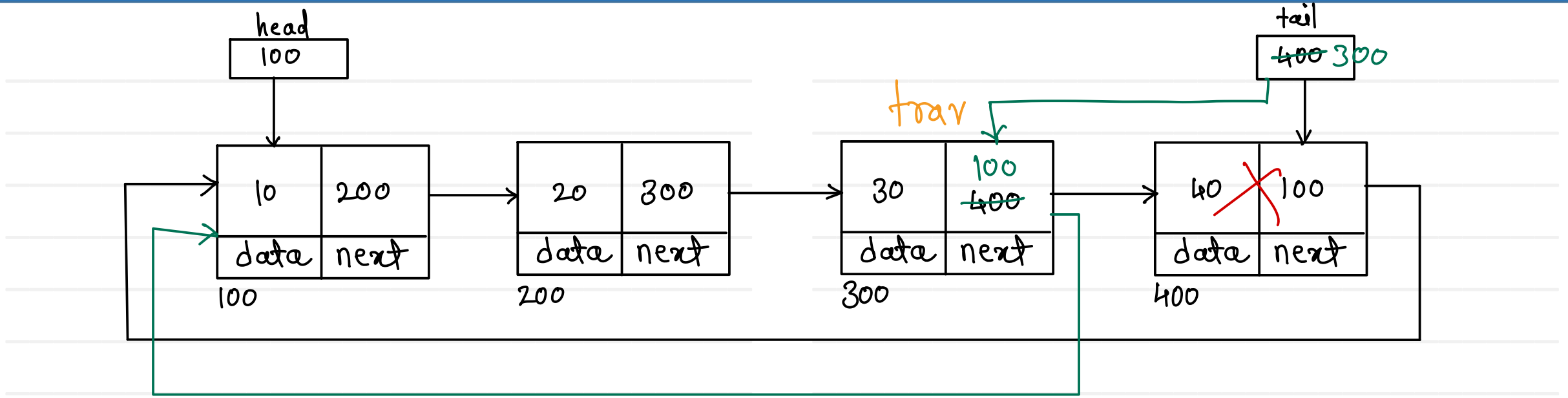


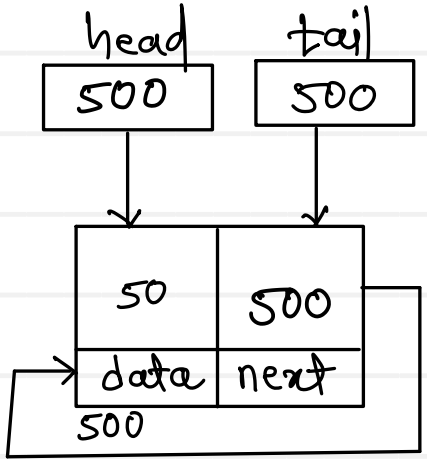

```
tail.next = head.next
head = head.next
```

```
newnode.next = head
tail.next = newnode
head = newnode
```

```
tail = newnode
```

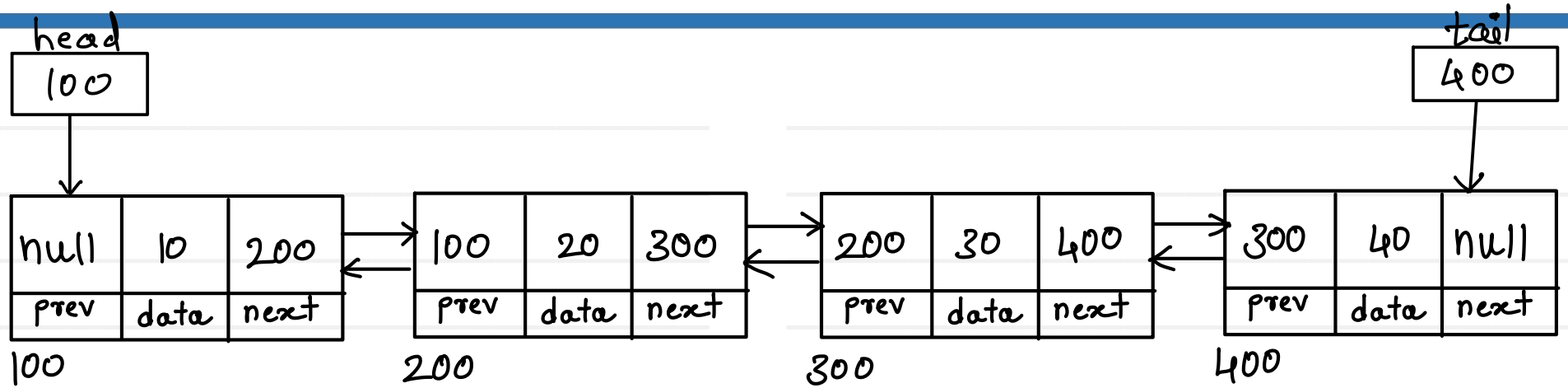
Singly Circular Linked List - Delete last






```
Node trav = head;
while( trav.next.next != head )
    trav = trav.next;
```

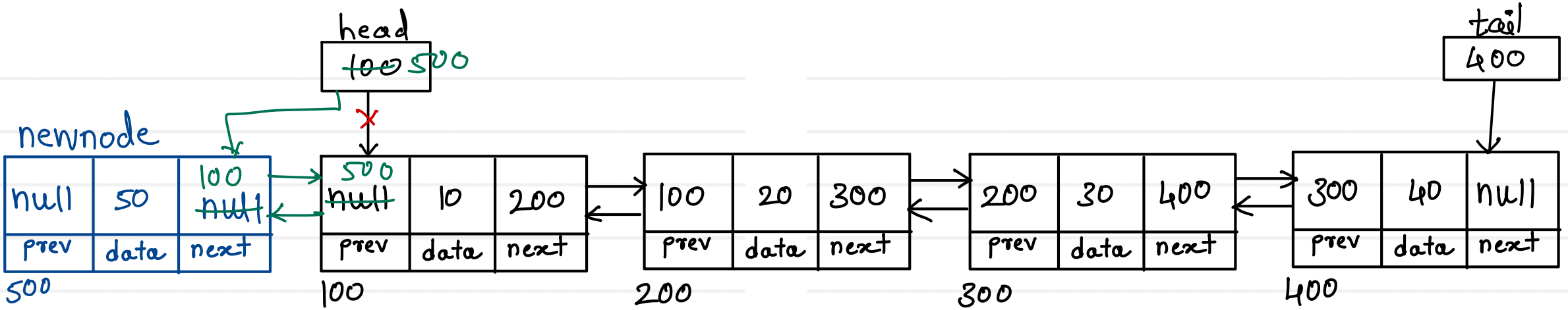
Doubly Linear Linked List - Display

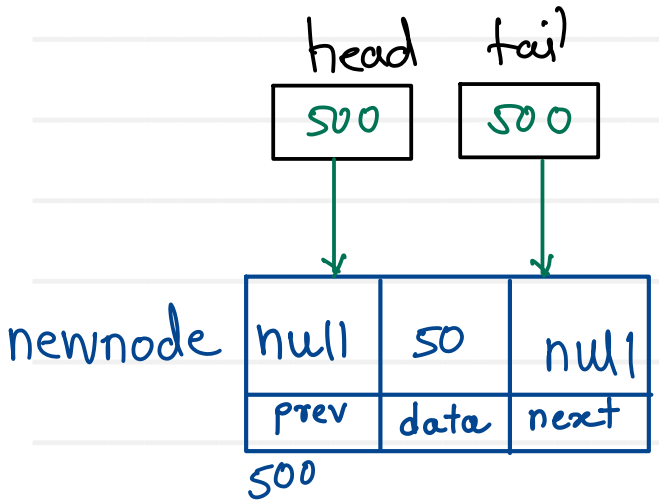


1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

1. Create trav & start at last node
2. print current node
3. go on prev node
4. repeat step 2 & 3 till first node

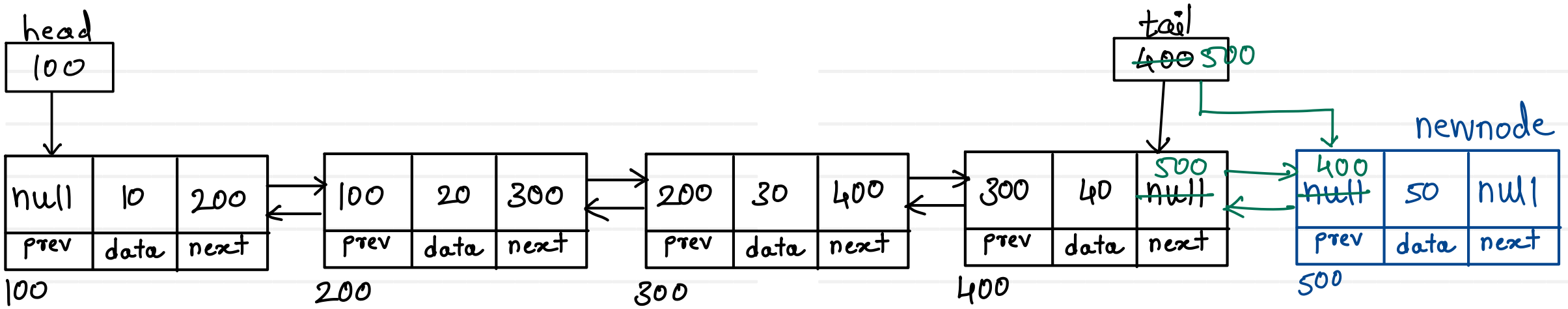
Doubly Linear Linked List - Add first

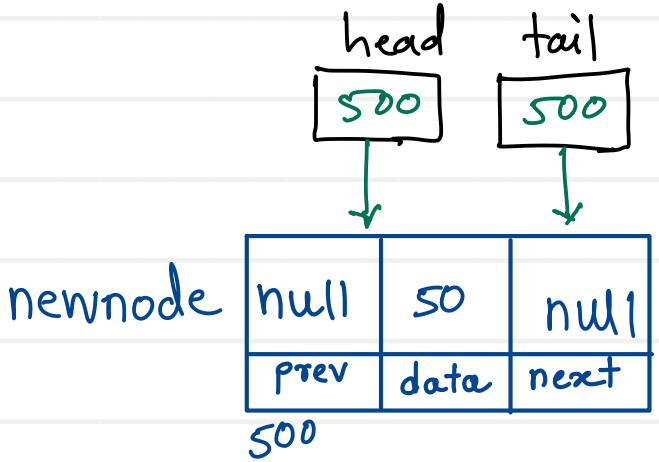




0. create a newnode
1. if list is empty, then add newnode into head & tail
2. if list is not empty,
   a. add first node into next of newnode
   b. add newnode into prev of first node
   c. move head on newnode

Doubly Linear Linked List - Add last

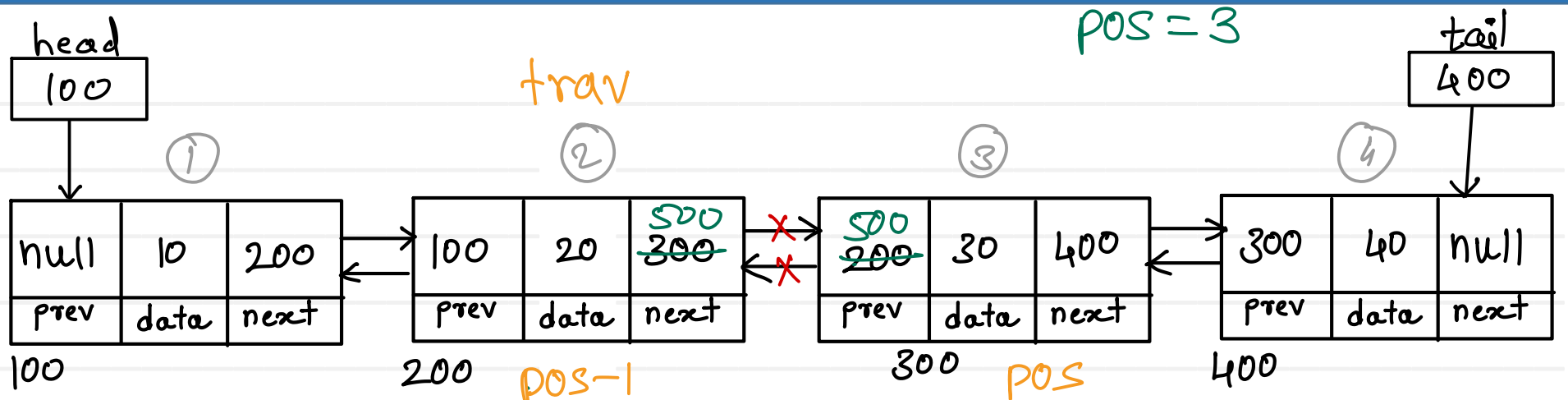




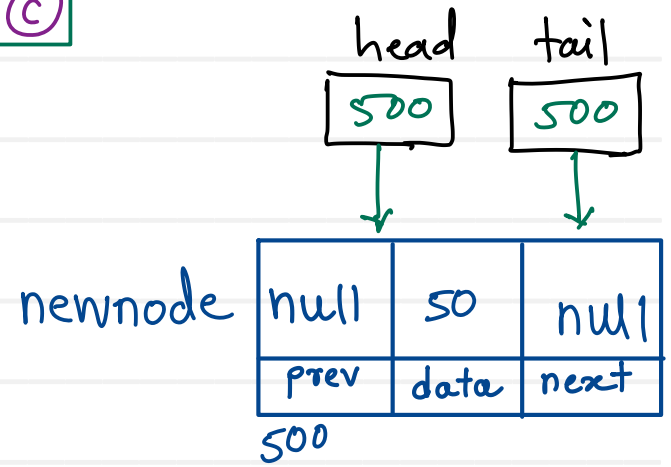
0. create a newnode
1. if list is empty, then add newnode into head & tail
2. if list is not empty,
   a. add last node into prev of newnode
   b. add newnode into next of last node
   c. move tail on newnode

Doubly Linear Linked List - Add position

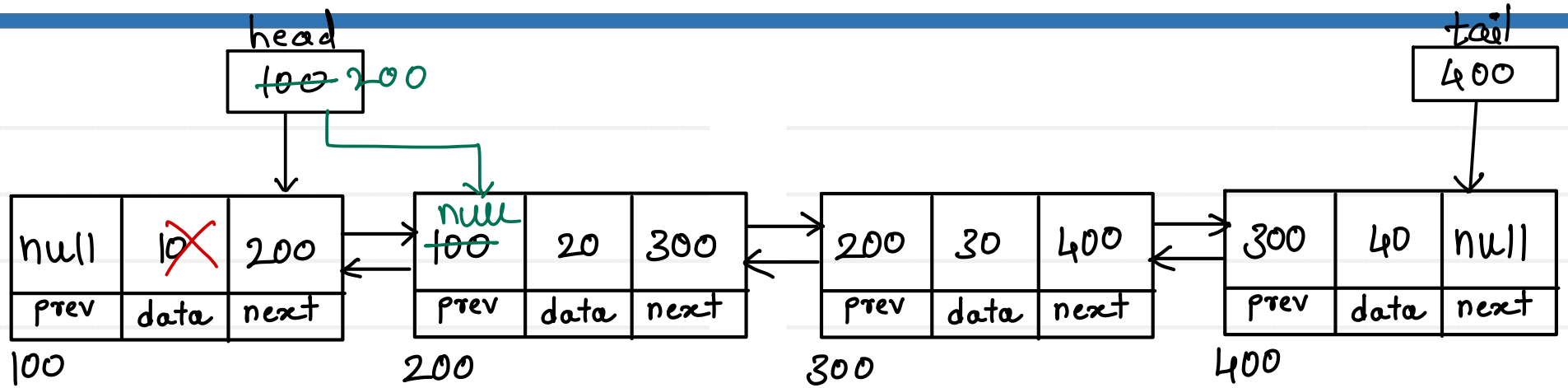
size = 4
valid positions:
1 → size+1
invalid positions:
< 1
> size+1

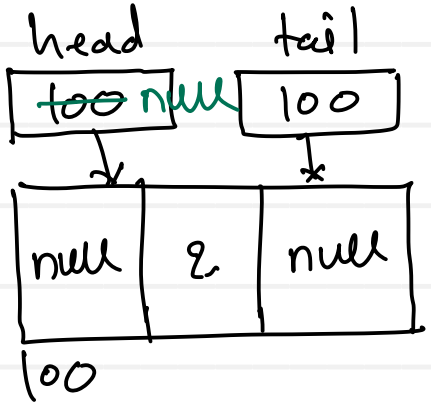


traverse till pos-1 node
a. add pos node into next of newnode
b. add pos-1 node into prev of newnode
c. add newnode into prev of pos node
d. add newnode into next of pos-1 node



Doubly Linear Linked List - Delete first

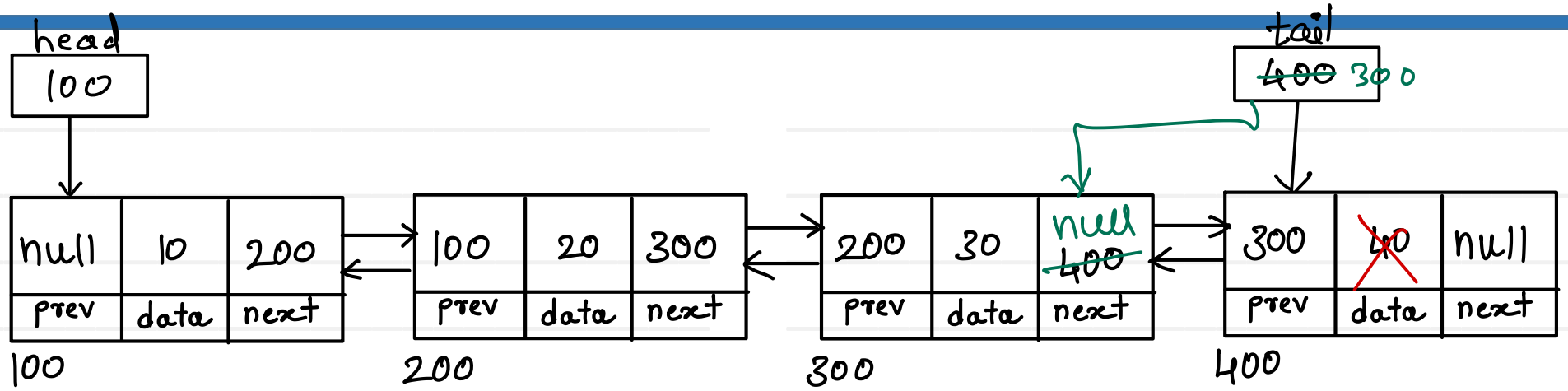




head = head.next
head.prev = null

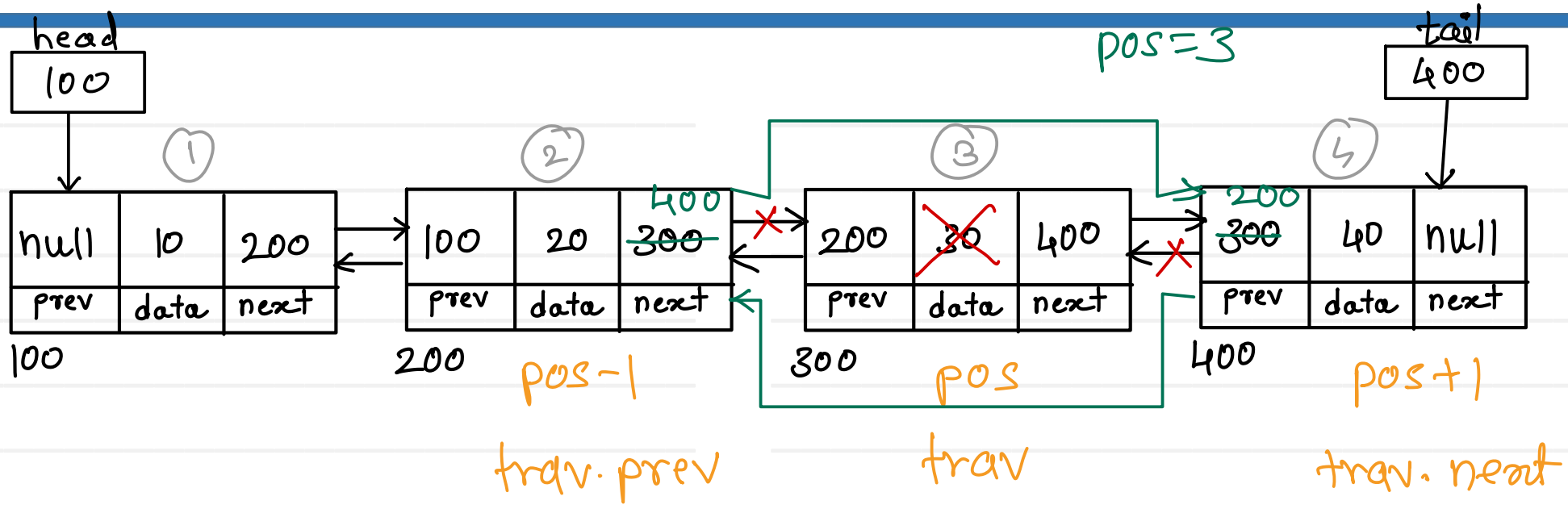
1. if list is empty, return
2. if single node, release it
3. if multiple node
   a. move head on second node
   b. add null into prev. of second node

Doubly Linear Linked List - Delete last



1. if list is empty, return
2. if single node, release it
3. if multiple node
   a. move tail on second last node
   b. add null into next of second last node

Doubly Linear Linked List - Delete Position



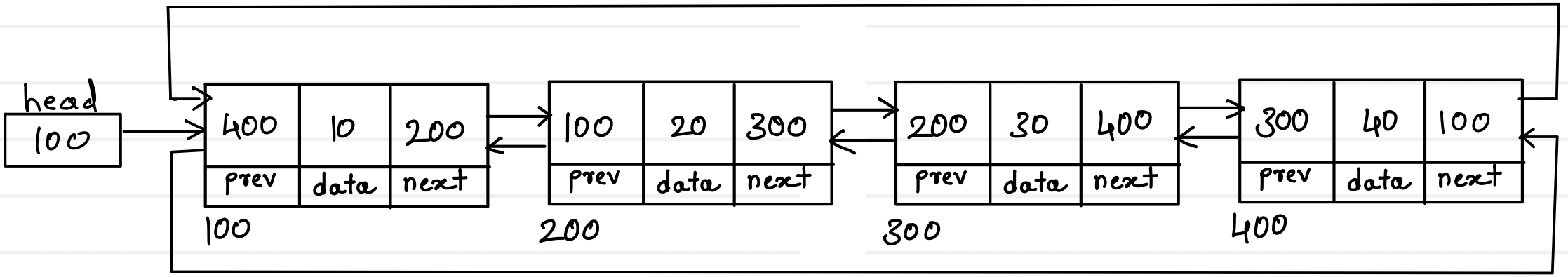
size = 4

valid positions :
1 to size

invalid positions :
< 1
> size

a. traverse till pos node
b. add pos+1 node into next of pos-1 node
c. add pos-1 node into prev of pos+1 node

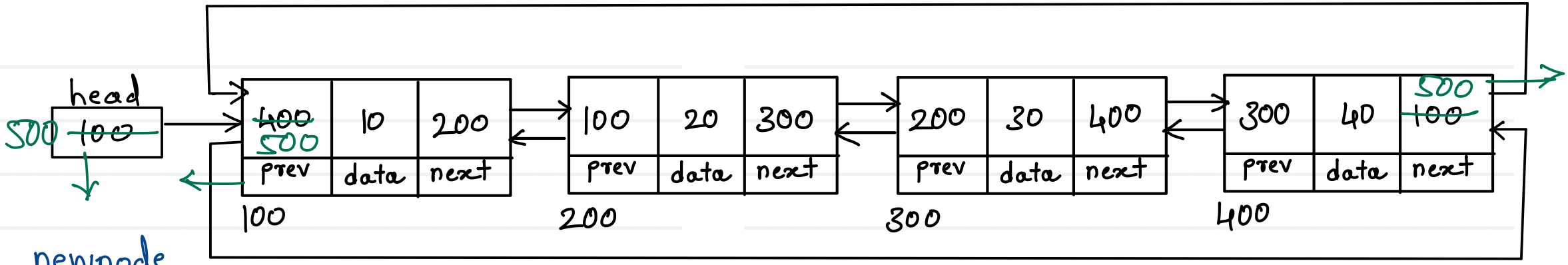
Doubly Circular Linked List - Display

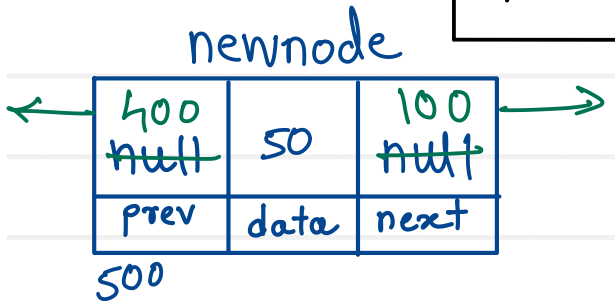


1. Create trav & start at first node
2. print current node data
3. go on next node
4. repeat step 2 & 3 till last node

1. Create trav & start at last node
2. print current node
3. go on prev node
4. repeat step 2 & 3 till first node

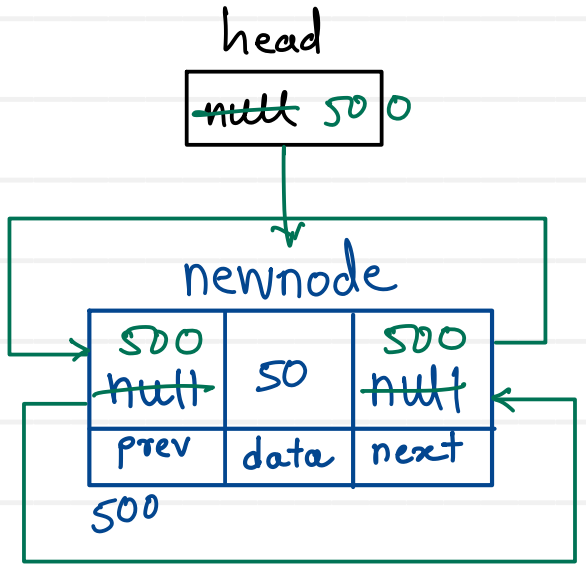
Doubly Circular Linked List - Add first



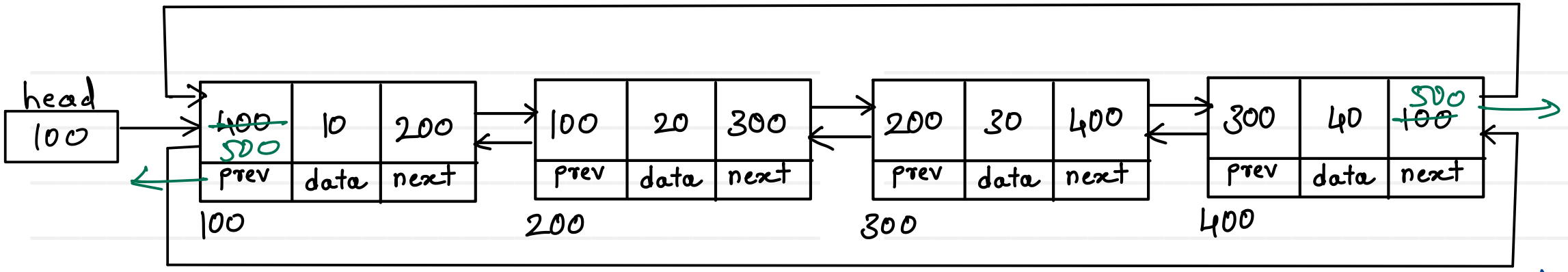


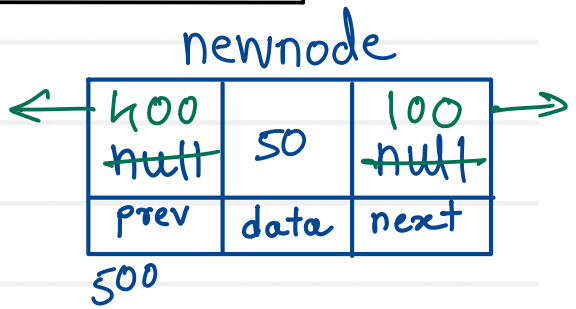
1. newnode.next = first node
2. newnode.prev = last node
3. next of last node = newnode
4. prev of first = newnode
5. move head on newnode

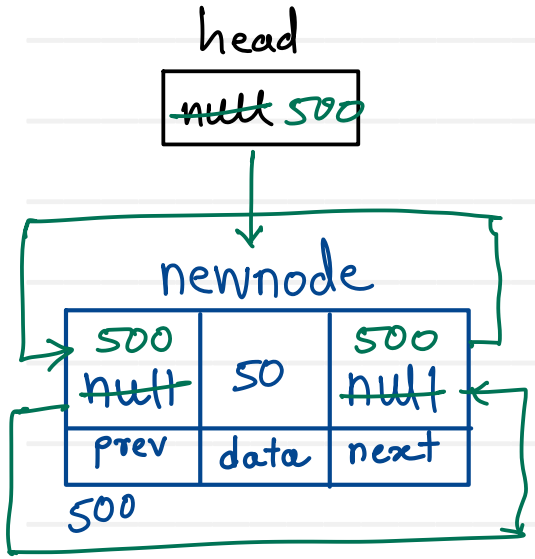
head = newnode,
newnode.next = newnode
newnode.prev = newnode



Doubly Circular Linked List - Add last

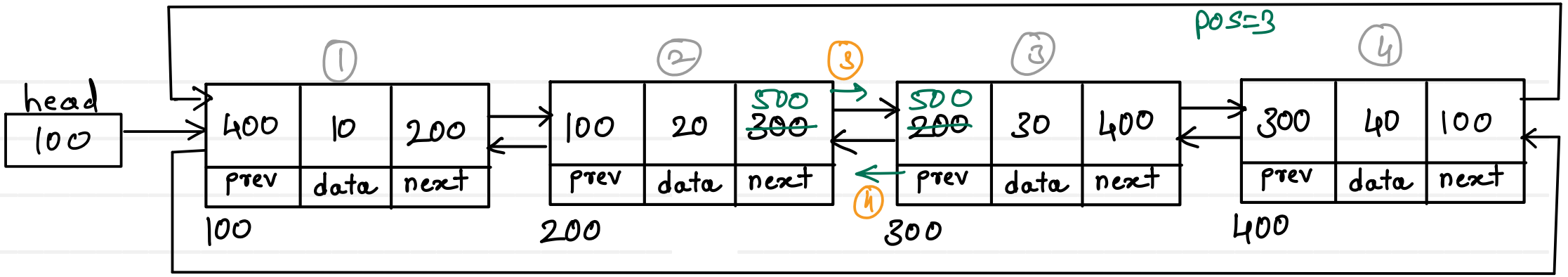




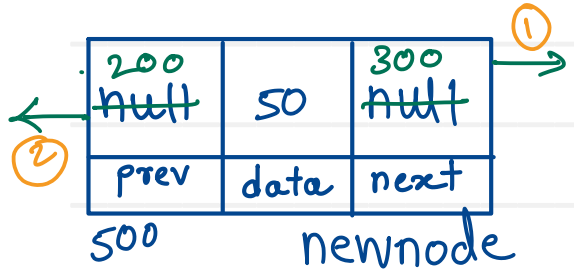


1. newnode.next = first node
2. newnode.prev = last node
3. next of last node = newnode
4. prev of first = newnode

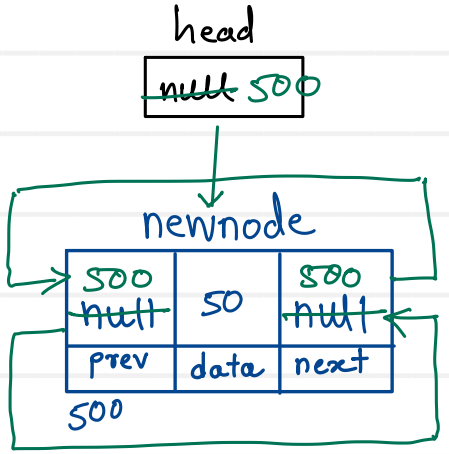
Doubly Circular Linked List - Add position



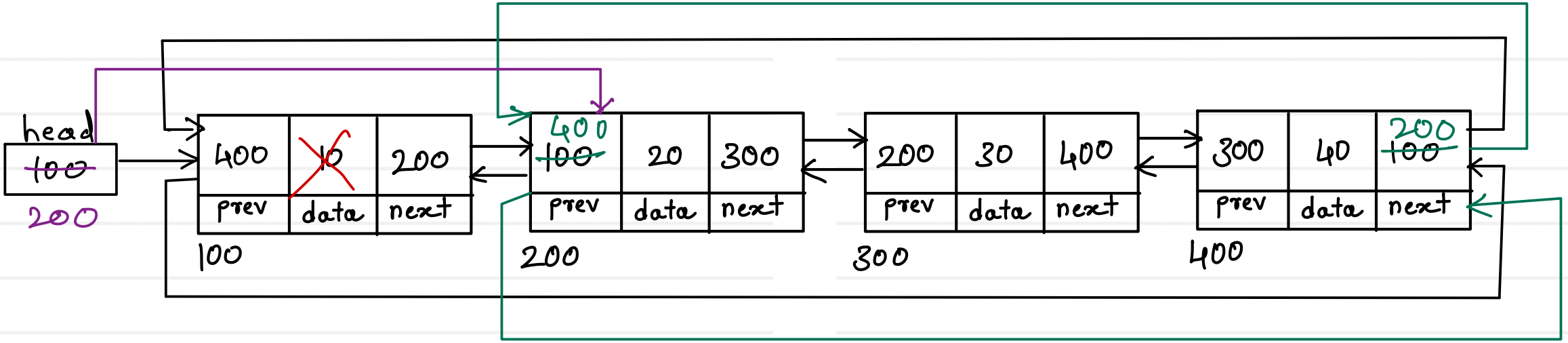
1. create node
2. if list is empty
   a. add newnode into head
   b. make list circular
3. if list is not empty.
   a. traverse till pos-1 node
   b. add pos node into next of newnode
   c. add pos-1 node into prev of newnode
   d. add newnode into next of pos-1 node
   e. add newnode into prev of pos node



$T(n) = O(1)$

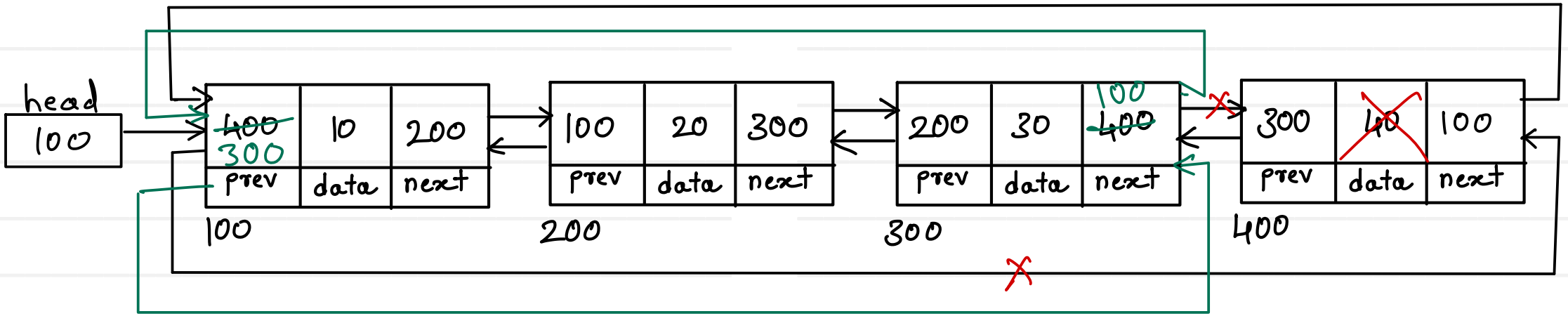


Doubly Circular Linked List - Delete first



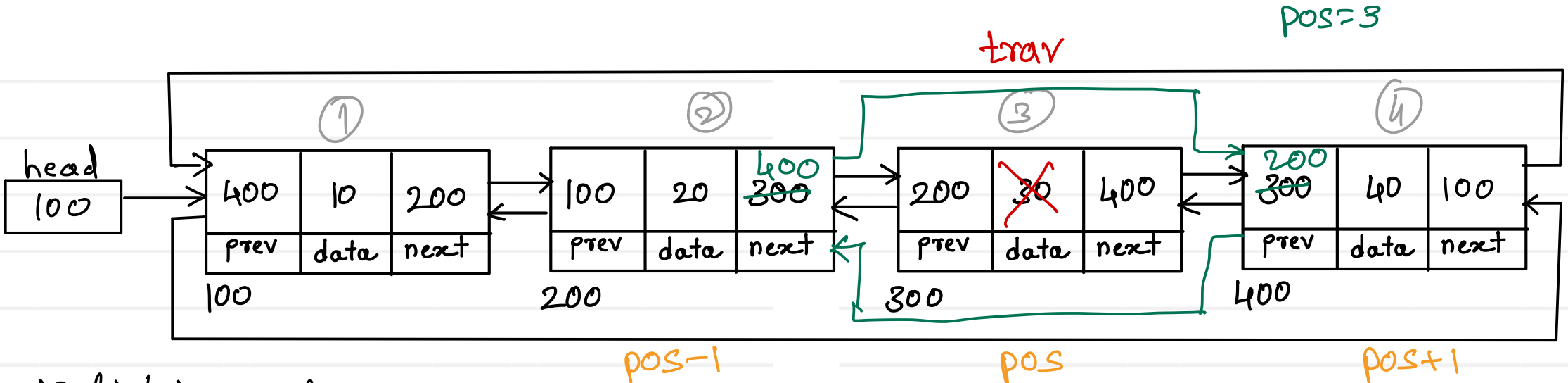
1. add second node into next of last node
2. add last node into prev of second node
3. move head on second node

Doubly Circular Linked List - Delete last



1. add second last node into prev of first node
2. add first node into next of second last node

Doubly Circular Linked List - Delete position



1. if list is empty
   return;
2. if list has single node
   head = tail = null;
3. if list has multiple nodes
   a. traverse till pos node
   b. add pos+1 node into next of pos-1 node
   c. add pos-1 node into prev of pos+1 node

trav → pos
trav.prev → pos-1
trav.next → pos+1

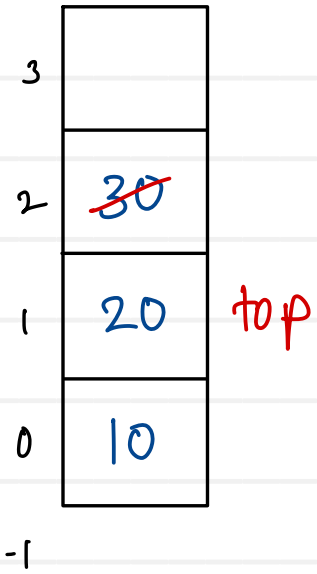
$T(n) = O(n)$

Stack

- Stack is a linear data structure which has only one end - top
- Data is inserted and removed from top end only.
- Stack works on principle of “Last In First Out” / “LIFO”

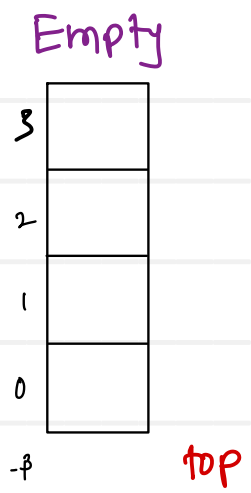
capacity = 4

top always points to last inserted data

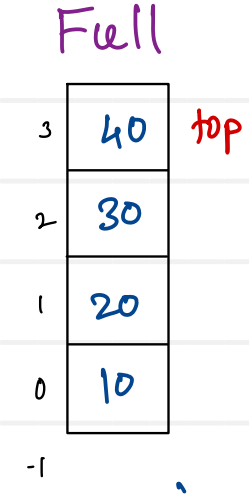


Operations :

1. Push :
   a. reposition top (top++)
   b. add data/value at top index
2. Pop :
   a. reposition top (top--)
3. Peek :
   a. read data/value of top index



top == -1



top == size-1

Using linked list :

1. Add first / Delete first
2. Add last / Delete last

Empty : head == null

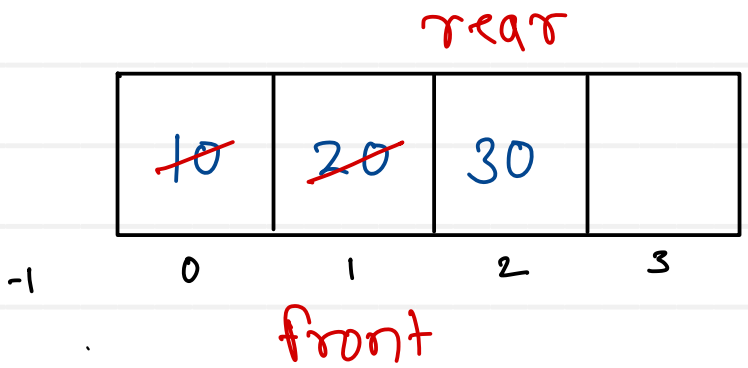
Full : never be full

Linear queue

- linear data structure which has two ends - front and rear
- Data is inserted from rear end and removed from front end
- Queue works on the principle of “First In First Out” / “FIFO”

Using linked list:
① Add first / delete last
② Add last / delete first
Empty: head == null
Full : never be full.

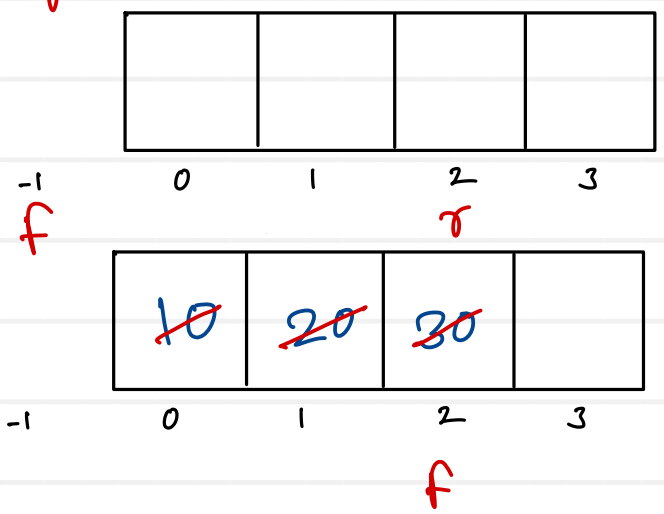
capacity = 4



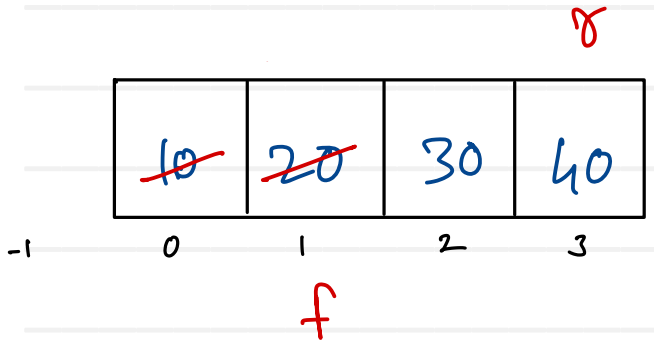
Operations:
1. Push/enqueue:
   a. reposition rear (++)
   b. add data/value at rear index
2. Pop/dequeue:
   a. reposition front (++)
3. Peek:
   a. read data from front end (front+1)

Linear queue - Conditions

Empty

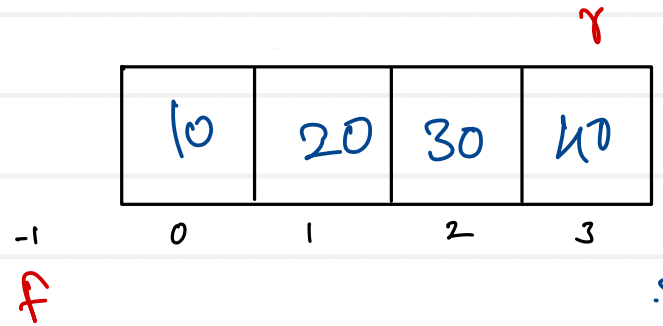


front == rear



- if linear queue is partially empty then empty locations can not be reused again, this leads to poor memory utilization.

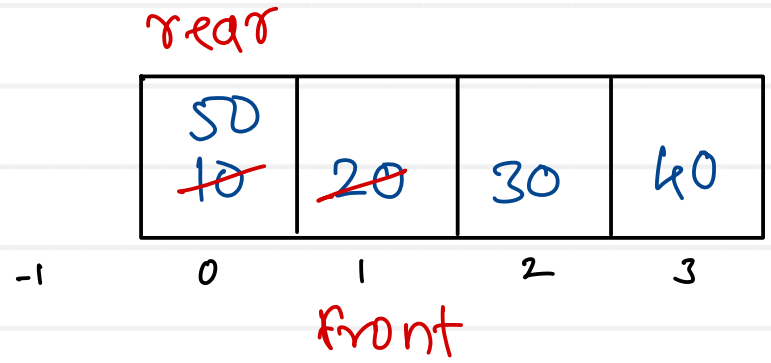
Full



rear == size-1

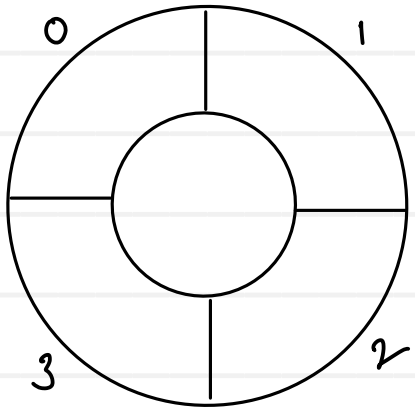
Circular queue

Capacity=4



Operations :

1. Push / enqueue / add / insert :
   a. reposition rear (inc)
   b. add value at rear index
2. Pop / dequeue / delete / remove :
   a. reposition front (inc)
3. Peek :
   a. read / return data of front+1 index



front = (front+1) % size

rear = (rear+1) % size

front = rear = -1

= (-1+1) % 4 = 0

= (0+1) % 4 = 1

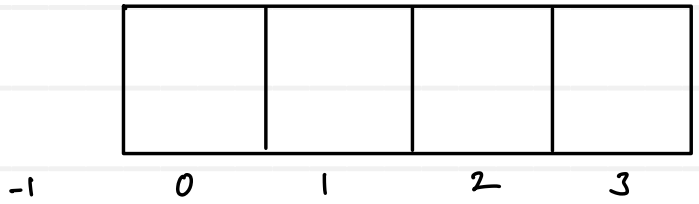
= (1+1) % 4 = 2

= (2+1) % 4 = 3

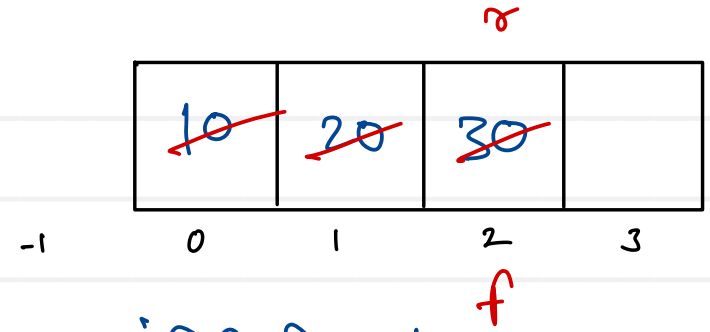
= (3+1) % 4 = 0

Circular queue - Conditions

Empty

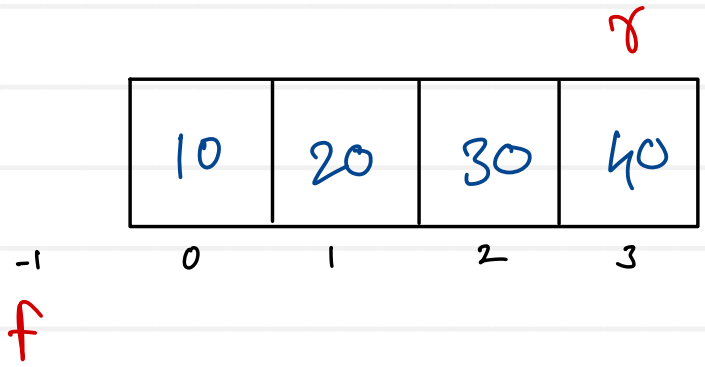


front == rear && rear == -1

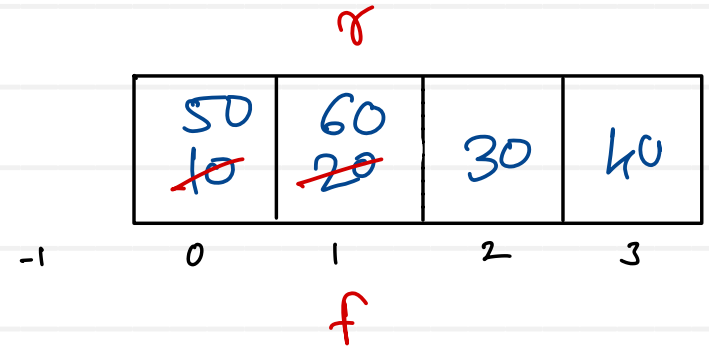


if(front == rear)
front = rear = -1

Full



(front == -1 && rear == size-1)



|| (front == rear && rear != -1)

Ascending stack

top = -1

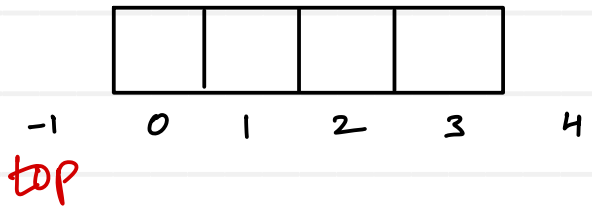
push : top++
arr[top] = value

pop : top--

peek : arr[top]

Empty : top == -1

Full : top == size-1



Descending stack

top = size

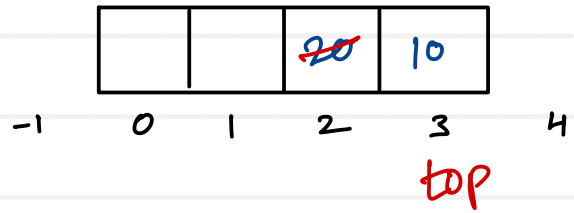
push : top--
arr[top] = value

pop : top++

peek : arr[top]

Empty : top == size

Full : top == 0



Array Vs Linked list

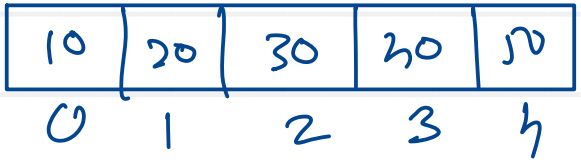
Array

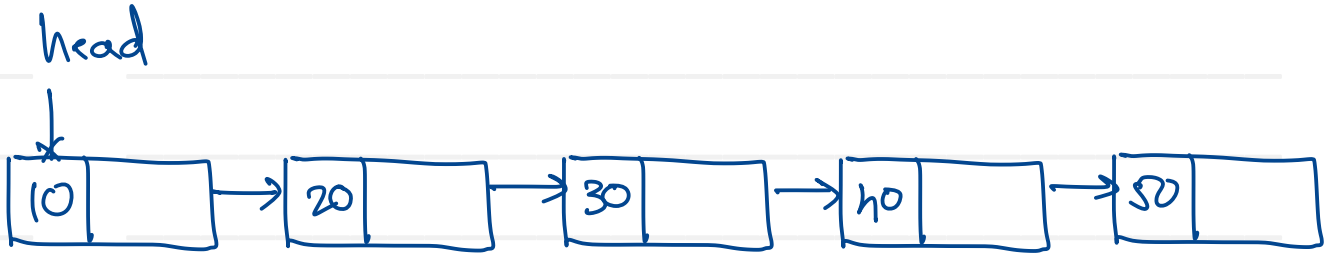
- Array space inside memory is continuous
- Array can not grow or shrink at runtime
- Random access of elements is allowed
- Insert or delete, needs shifting of array elements
- Array needs less space

Linked list

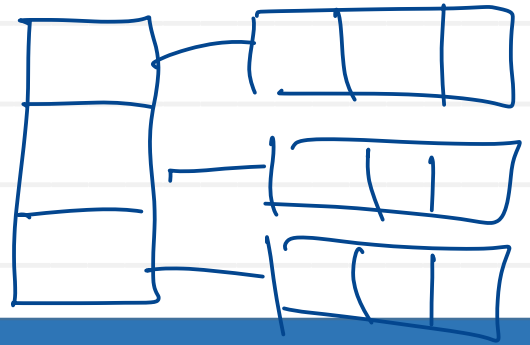
- Linked list space inside memory is not continuous
- Linked list can grow or shrink at runtime
- Random access of elements is not allowed
- Insert or delete, need not shifting of linked list elements
- Linked list needs more space

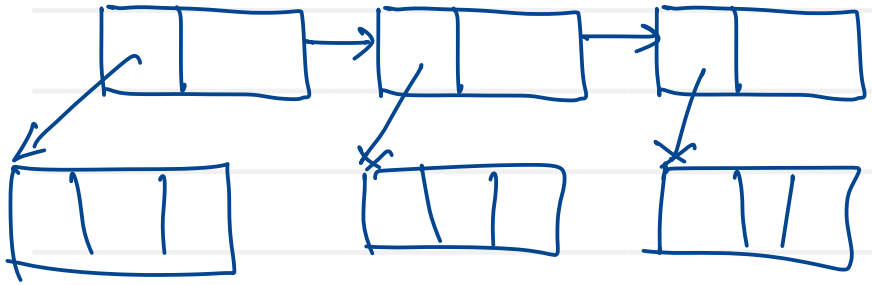
```
int arr[5];
```






```
class student {
        int rollno;
          :
};
student arr[] = new ... ;
```





Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com