Types of Linked List - Part II

Course on C-Programming & Data Structures: GATE - 2024 & 2025

Vishvadeep Gothi • Lesson 30 • Jan 5, 2023

Data Structure:
Header & Doubly Linked List

By: Vishvadeep Gothi

Hello!

I am Vishvadeep Gothi

I am here because I love to teach

Application of Linked List

- To store polynomials
  - Univariate
  - Bivariate

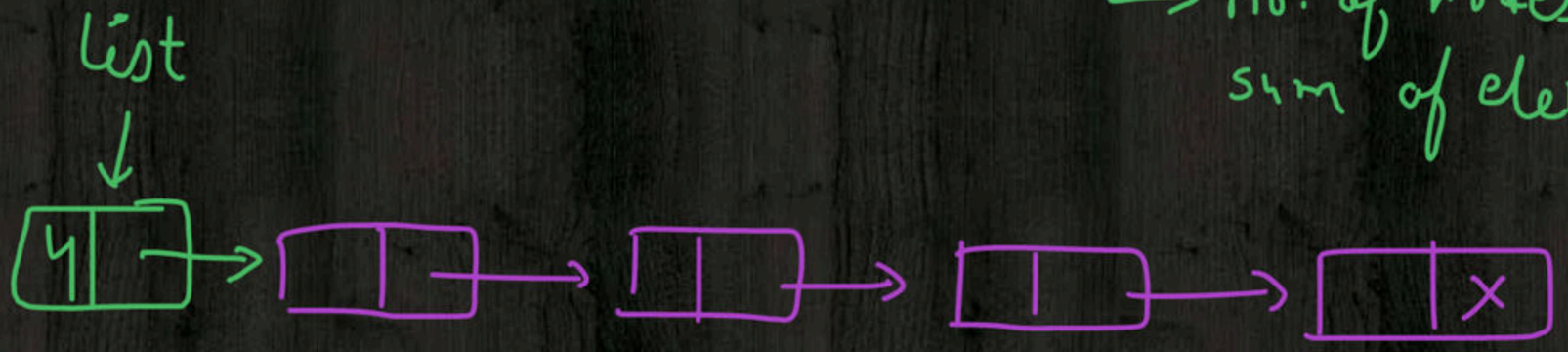
Disadvantage of Singly Linked List

- The link part of last node is not utilized → soln => circular
- The address of predecessor is not known
- Stepping backward is not possible

→ soln => doubly linked

Header List

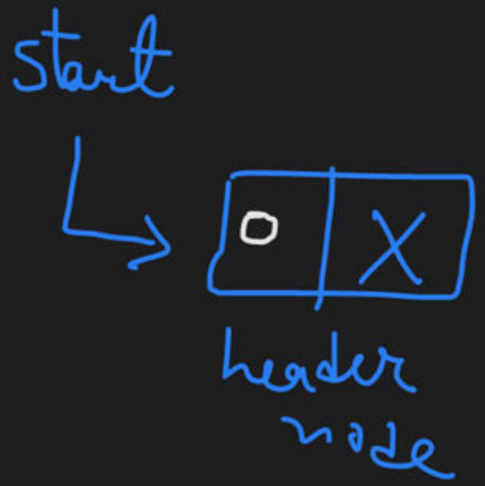
- Contains special first node called **Header node**.
- Header node contains some summery information.
  - ↳ no. of nodes, sum of elements




```
(list -> data)++ ;
(list -> data)-- ;
```

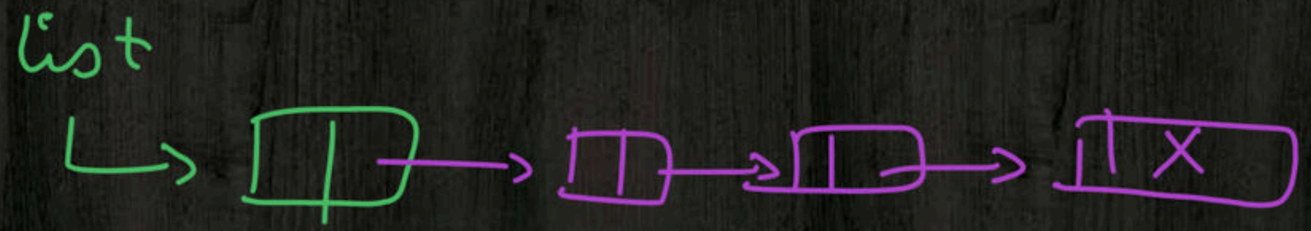
Headerlist :-

Empty

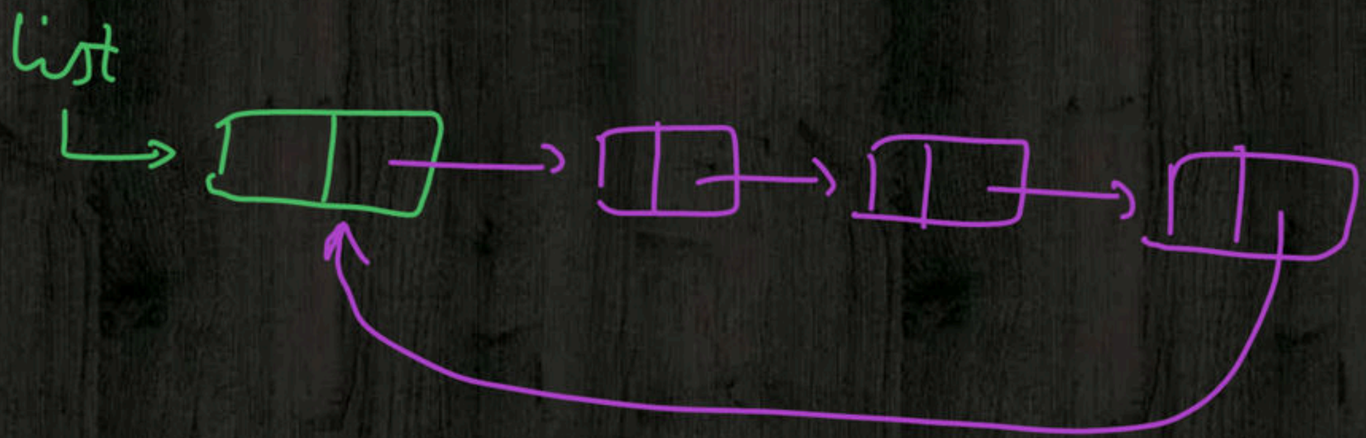


Types of Header List

1. Grounded



2. Circular



Traversal in headere list

grounded

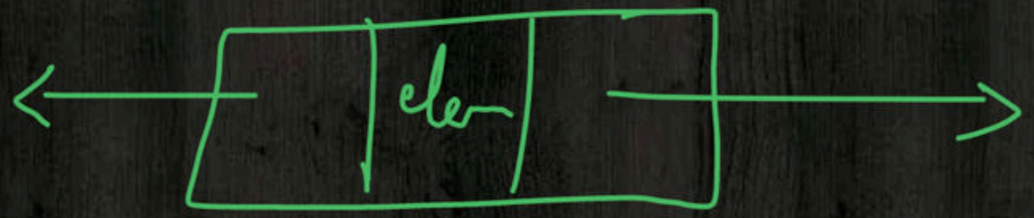
```
struct node *p = start -> link
while (p)   or   while (p != NULL)
{
 process p -> data
 p = p -> link
}
```

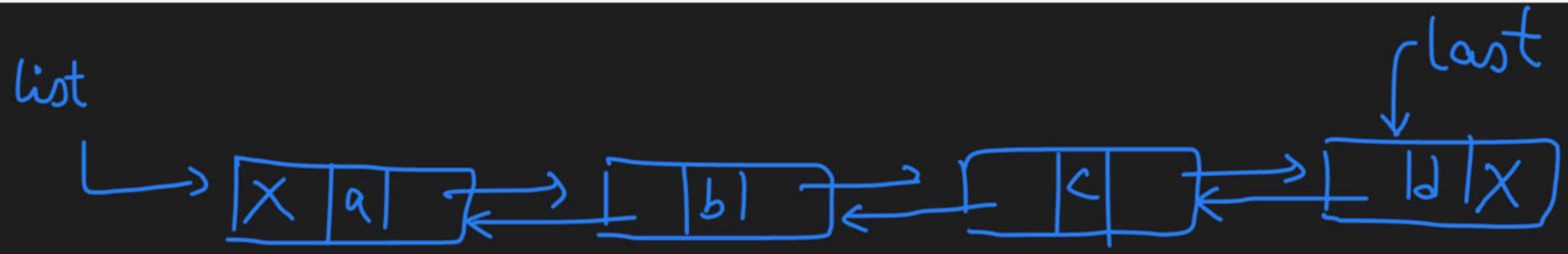
circular

```
struct node *p = start -> link
while (p != start)
{
 process p -> data
 p = p -> link
}
```

Doubly Linked List

every node stores 2 addresses → ① add. of next node
→ ② add. of previous node





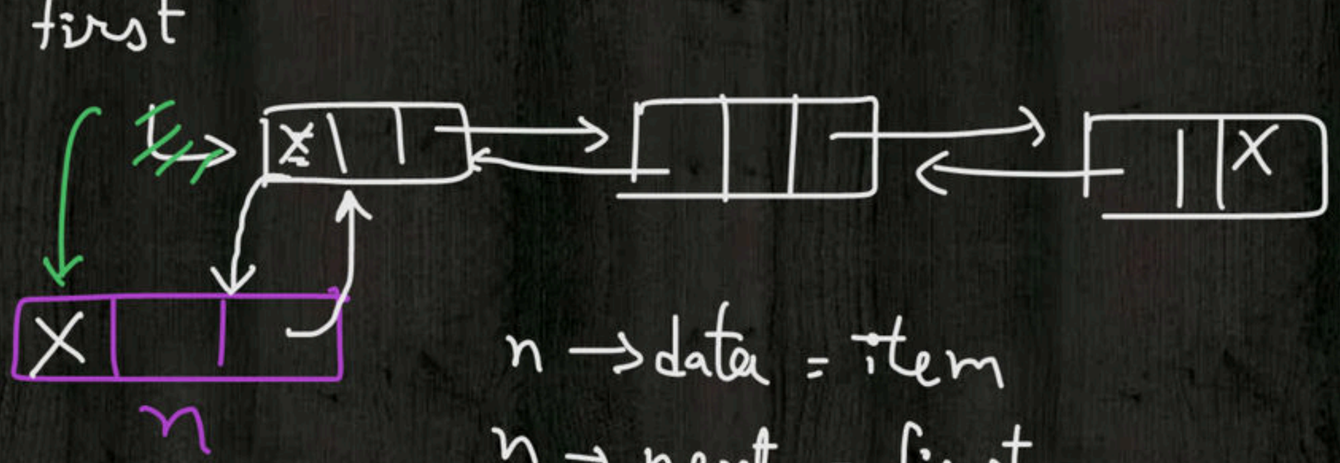

```
struct node
{
 char data;
 struct node *link;
};
```

for singly linked list

```
struct dnode
{
 char data;
 struct dnode *next;
 struct dnode *prev;
};
```

Insertion in Doubly Linked List

(1) Insertion at beginning :-



R T C. = $\theta(1)$

n → data = item

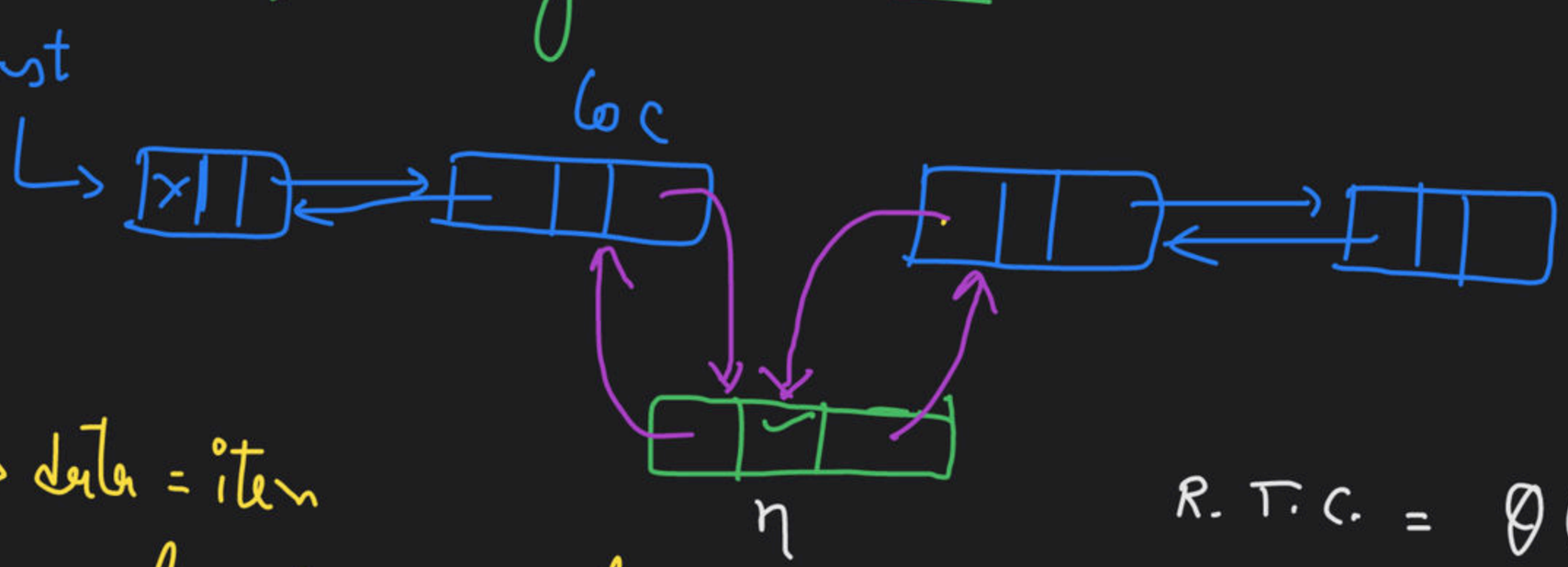
n → next = first

n → prev = NULL

first → prev = n or n → next → prev = n

first = n

② Insertion after a given node:-



n → data = item

n → next = loc → next

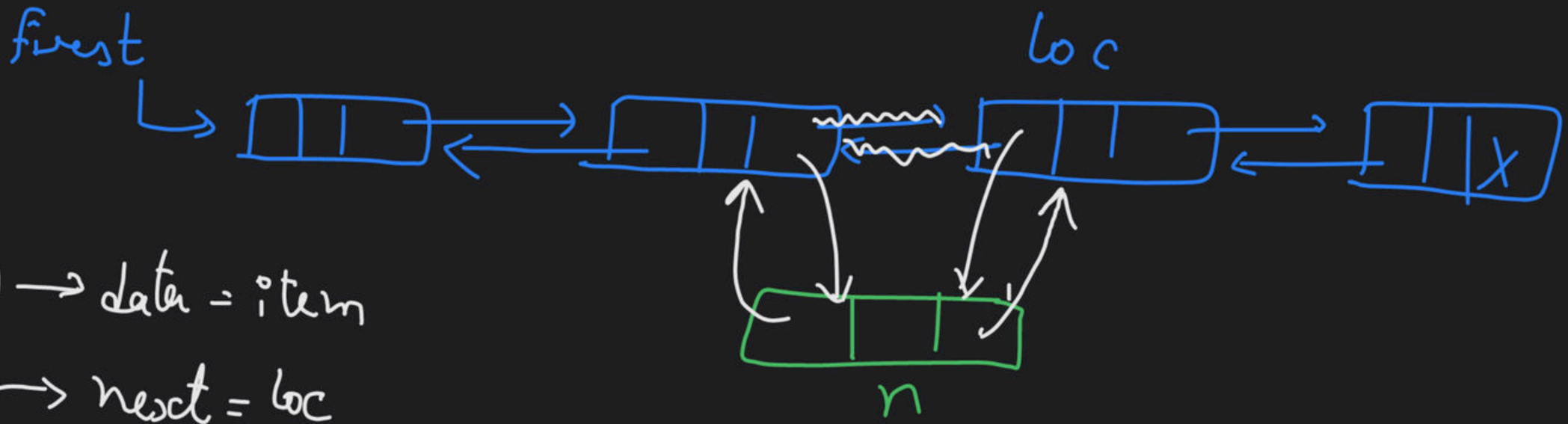
n → prev = loc

n → next → prev = n

loc → next = n

R. T. C. = $\theta(1)$

③ Insertion before given node



$n \rightarrow data = item$

$n \rightarrow next = loc$

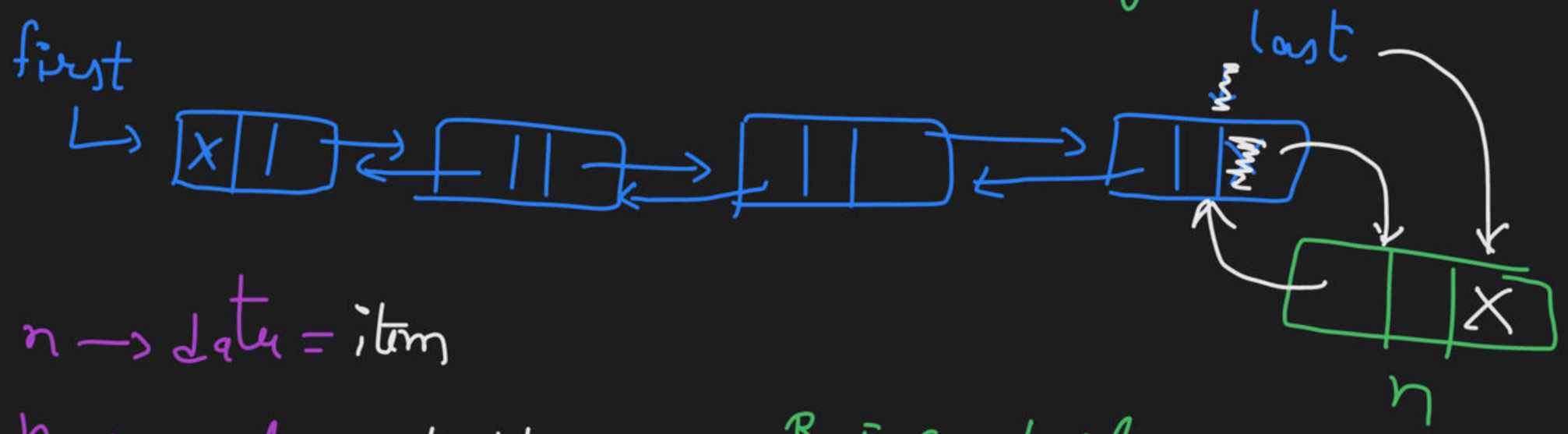
$n \rightarrow prev = loc \rightarrow prev$

$n \rightarrow prev \rightarrow next = n$

$loc \rightarrow prev = n$

RT complexity = $\theta(1)$

(4) Insertion at the end when add. of last node is given



n → data = item

n → next = NULL

n → prev = last

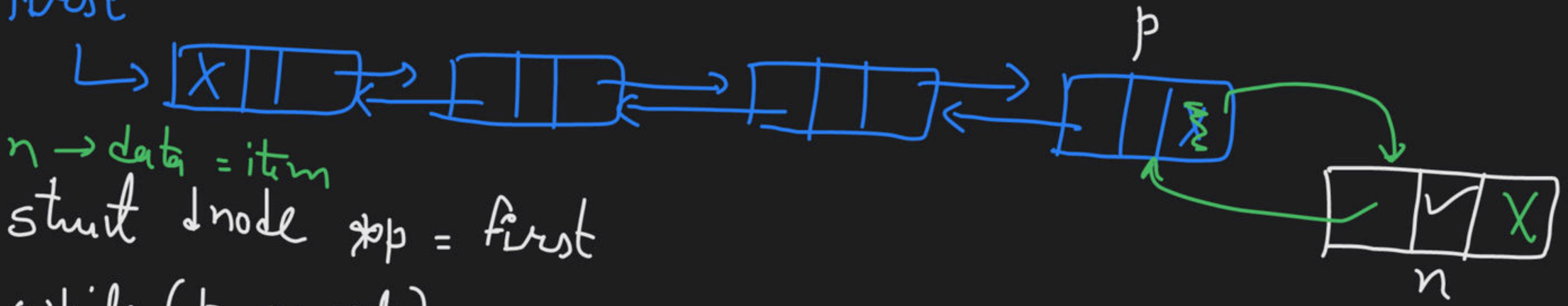
last → next = n

last = n

R.T. Complexity

= O(1)

③ Insertion at the end, when addr. of last node not given

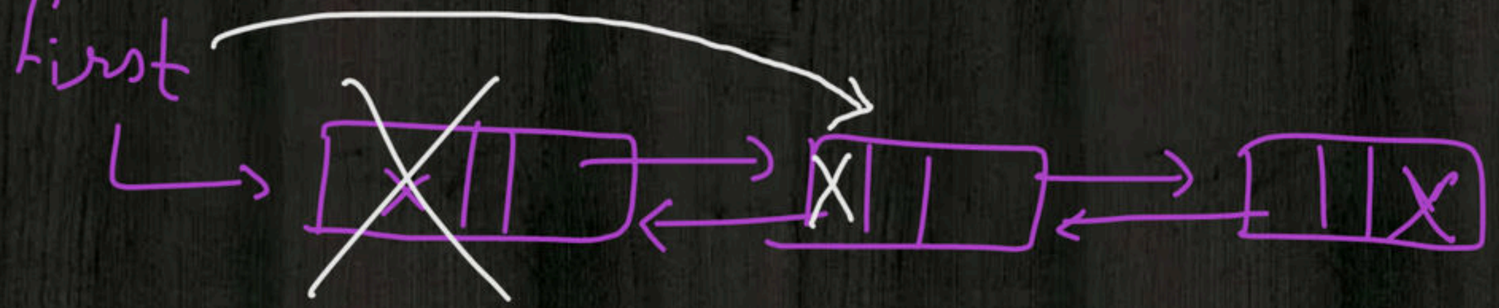



```
n → data = item
struct dnode *p = first
while (p → next)
{
  p = p → next
}
n → next = NULL
n → prev = p
p → next = n
```

R.T Complexity = $\Theta(n)$

Deletion in Doubly Linked List

① Deletion of first node :-

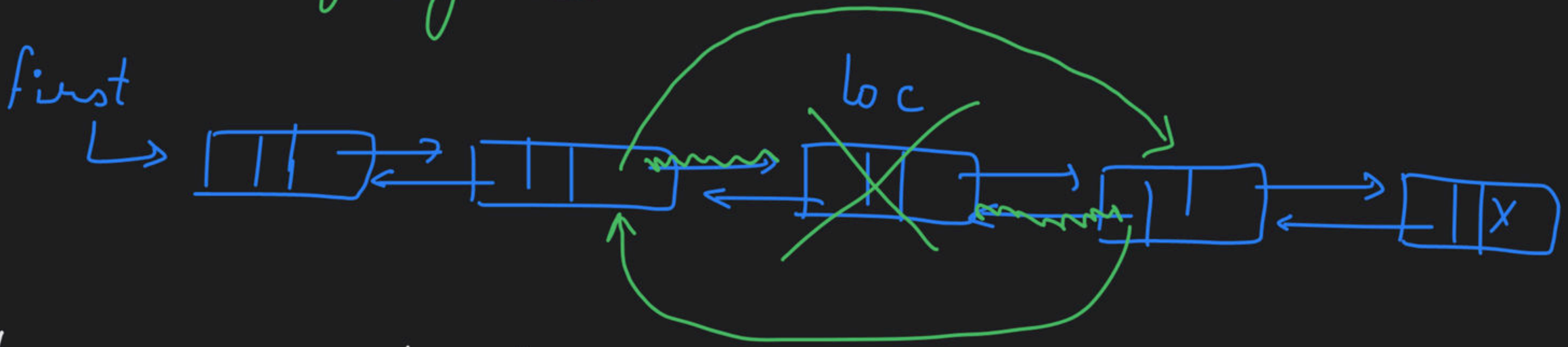


first = first → next

first → prev = NULL

R.T.C. = $\Theta(1)$

(2) Deletion of given node :-



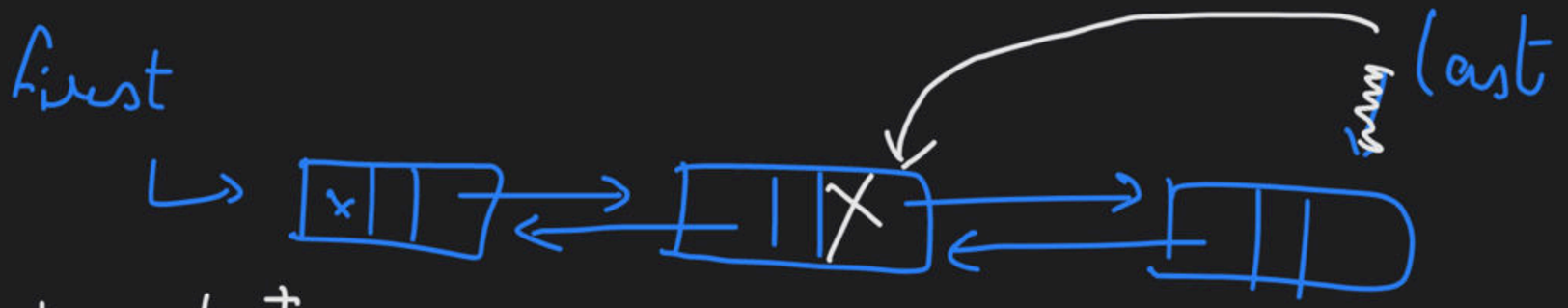
loc → prev → next = loc → next

loc → next → prev = loc → prev

free (loc)

R.T. Complexity = O(1)

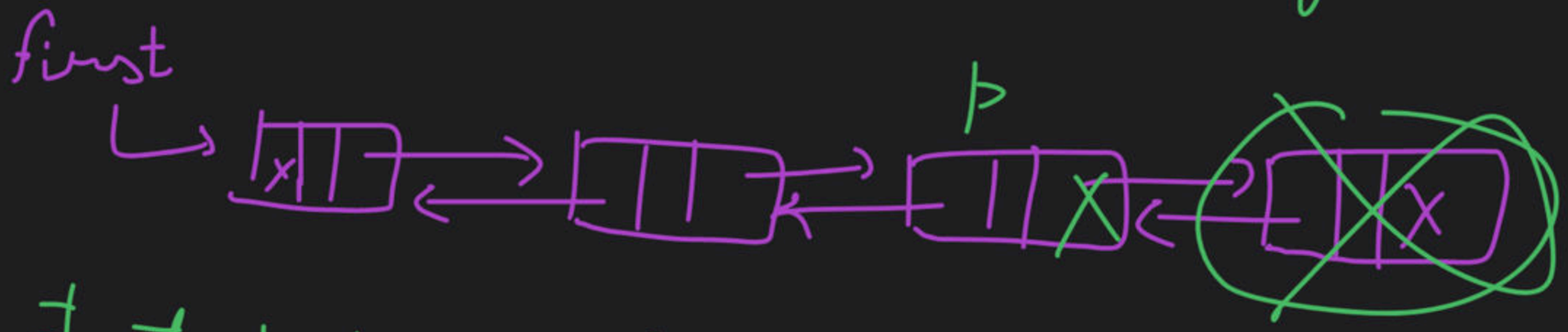
③ Deletion of last node, when add. of last node given:-




```
p = last
last = last -> prev;
last -> next = NULL;
free(p)
```

R.T.C = O(1)

④ Deletion of last node, when add. of last node not given



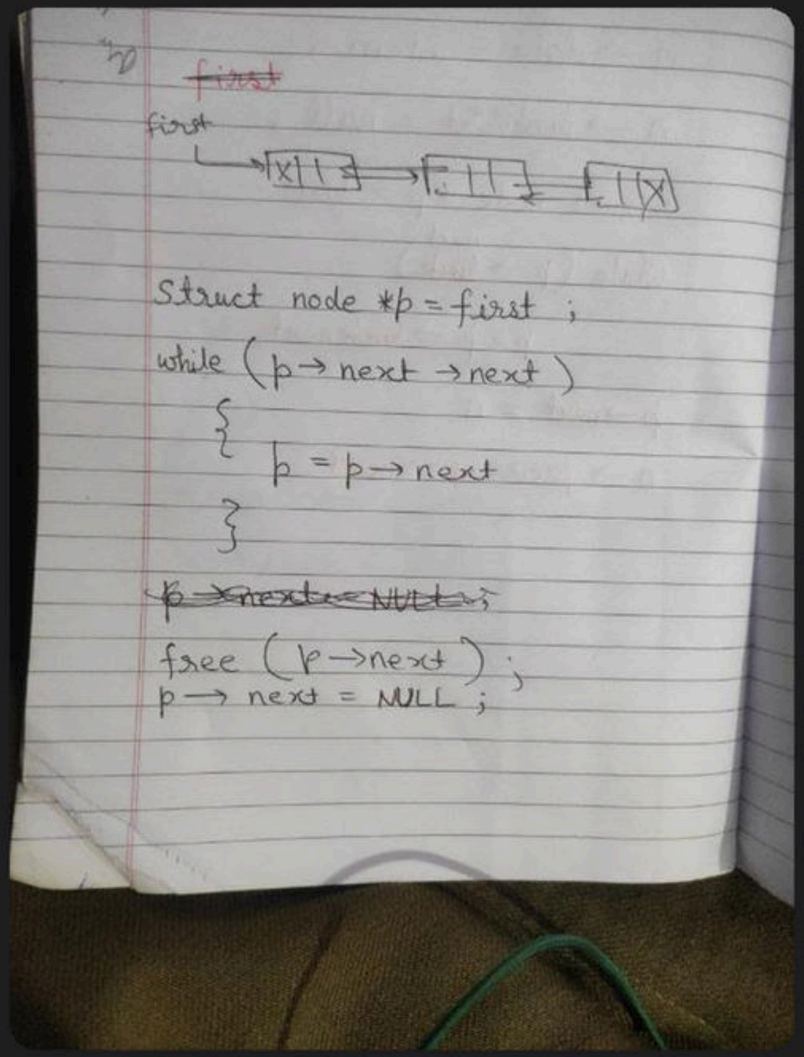

```
struct node *p = first
while (p → next → next)
{
  p = p → next
}
free (p → next)
p → next = NULL
```

R.T.C. = $\theta(n)$

unacademy

▲ 1 • Asked by Krishna Ku...

Please help me with this doubt



Question 1

- How many links updated when insertion and deletion done in doubly linked list?

	start	end	middle
Insertn	3	3	4
Deletn	1	1	2

Question 2

The following code is given to reverse the doubly linked list. Fill the blank?

```
void reverse(Dnode *head)
{
struct Dnode *p = head;
while(p ! = head)
{
swap(p → forw, p → back);
_____
}
swap( head → forw, head →  back)
}
```

Question 3

```
void fun(struct node * list)
{
  if(! list)
            return;

  fun(list → link);
  printf("%c", list → data);
}
```

What does the above function does on a singly linked list?

Question 4

What does fun() function do, if called as fun(NULL, First), where First is a list pointer, pointing to the first node of a singly list?

```
void fun(struct node * p, struct node *q)
{
  if(q)
    {
       fun(q, q → link);
       q → link =p;
    }
    else
        First = p;
}
```

(A) No changes in the list
(B) Reverses the list
(C) Swaps every consecutive pair of nodes
(D) Deletes last node and adds it to front

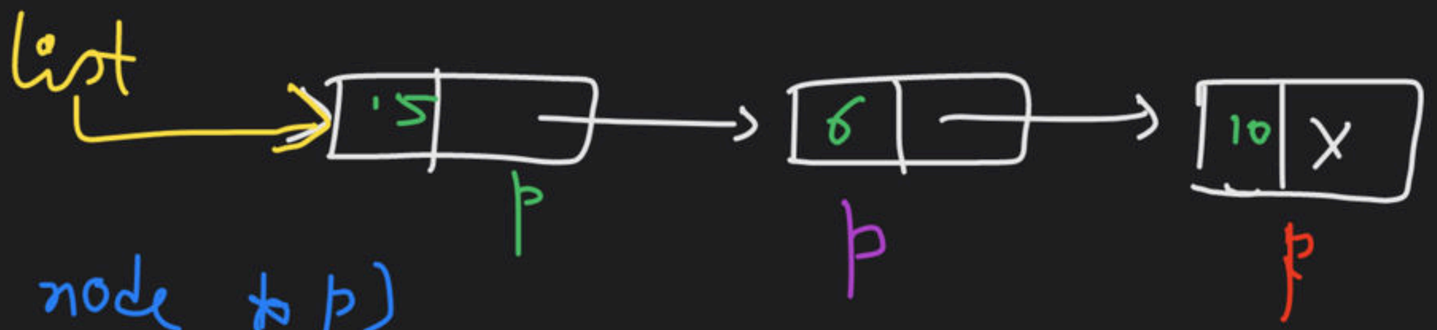
Question 5

What does fun() function do, if called with list pointers of 2 singly linked lists?

```
void fun(struct node * p, struct node *q)
{
struct Node *px, *qx;
if(!p)
   return q;
else if(!q)
   return p;
else
{
  px = p → link;
 qx = q → link;
 p → link = q;
 q → link = fun(px, qx);
 return p;
}
```

(A) Concatenation of 2 lists by selecting alternate nodes
(B) Append list p at the end of list q for all inputs
(C) Append list q at the end of list p for all inputs
(D) Append reverse of list p at the end of list q for all inputs

Ques)



p = NULL

```
struct node * fun ( struct node * p)
{
   if (!p)
     return ;
  printf("%d", p -> data);
  fun(p -> link);
}

fun (list)
```

15 6 10

struct node *

Question 6

```
[redacted] fun(struct node * list)
{
  if(! list → link)
              return list;

  return fun(list → link);
}
```

What does the above function does on a singly linked list?

Happy Learning