

# Linked list: Implementation

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

# Data Structure: Linked List 2

By: Vishvadeep Gothi



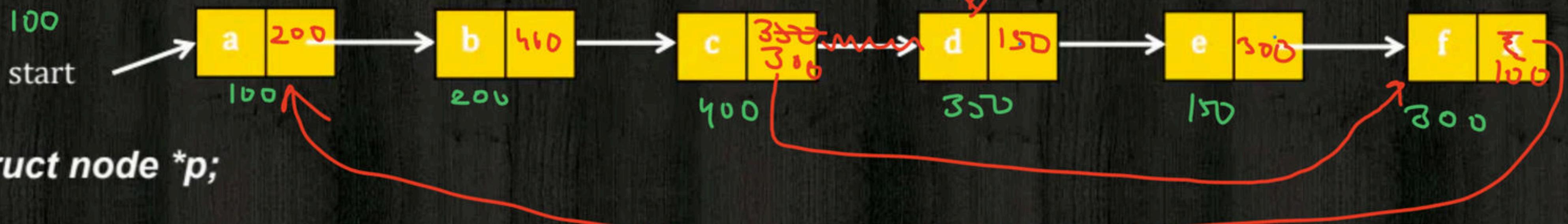
*Hello!*

# I am Vishvadeep Gothi

I am here because I love to teach

# Practice Question 1

What would be the output after the sequence of steps:



*struct node \*p;*

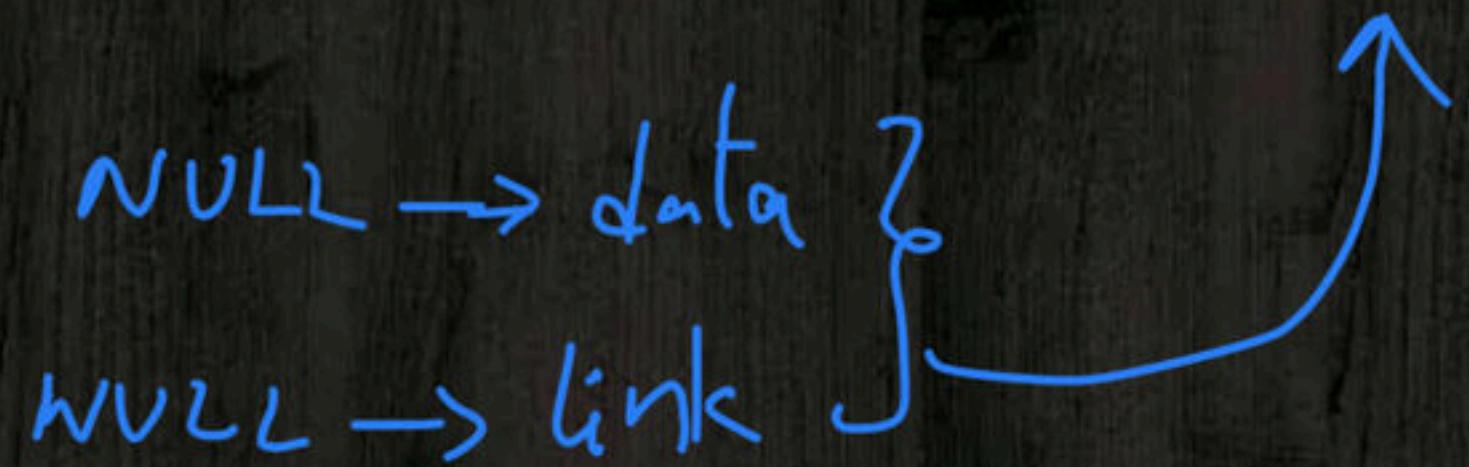
*p = start → link → link → link;*

*start → link → link → link = p → link → link;*

*p → link → link → link = start;*

*printf("%c", start → link → link → link → link → data); ↴*

# NULL Pointer Dereference

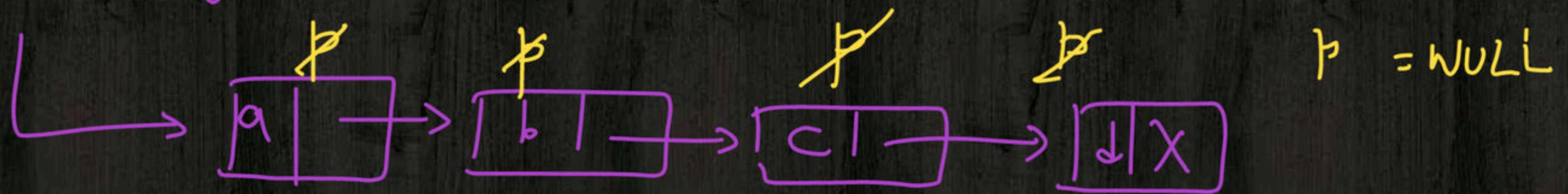


p = ..... j

if ( $p \rightarrow \text{link} \neq \text{NULL}$ )

# Traversing in Linked List

list / head / start



struct node \* $p$  = start;

while ( $p \neq \text{NULL}$ ) or while ( $p$ )  
{

process  $p \rightarrow \text{data}$  ;      R.T.

$p = p \rightarrow \text{link}$ ;

complexity =  $\Theta(n)$

}

# Number of Elements in Linked List

node

```
Struct node *p = start;  
int count = 0;  
while (p)  
{  
    count++;  
    p = p->link;  
}  
return count;
```

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

Consider a linked-list with integer elements. Return sum of all elements of list.

```
struct node *p = start;  
int sum = 0;  
while(p)  
{  
    sum = sum + p->data;  
    p = p->link;  
}
```

$O(n)$

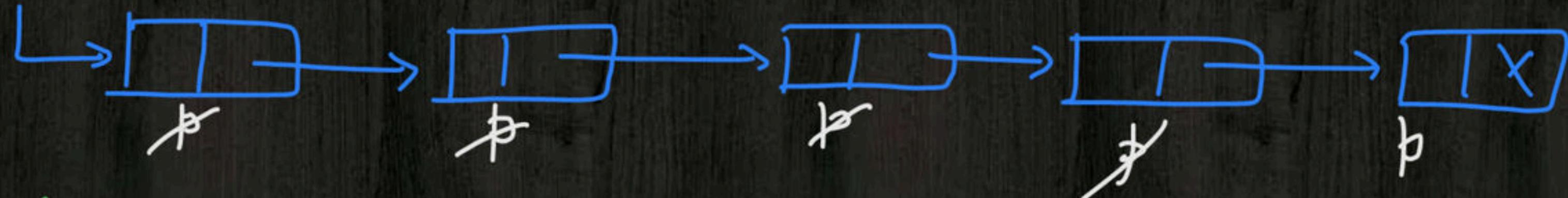
Find min of linked-list ?

```
if (start == NULL) return 0;  
int min = start->data;  
  
struct node *p = start->link;  
while (p)  
{  
    if (p->data < min)  
    {  
        min = p->data;  
    }  
    p = p->link;  
}  
return min;
```

$\Theta(n)$

# Address of Last Node in Linked List

start



start node  $\ast p = start;$

if ( $p == NULL$ ) return  $NULL$ ;

while ( $p \rightarrow link != NULL$ ) or while ( $p \rightarrow link$ )

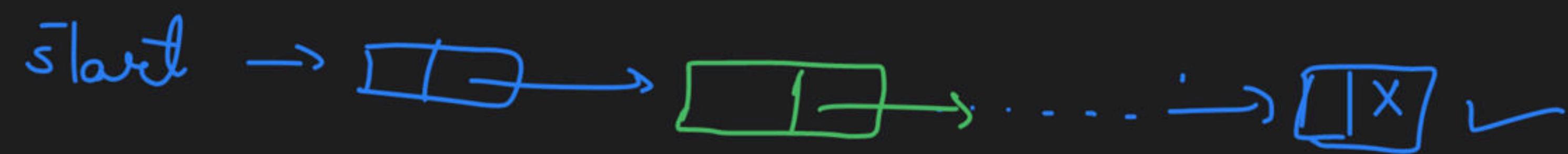
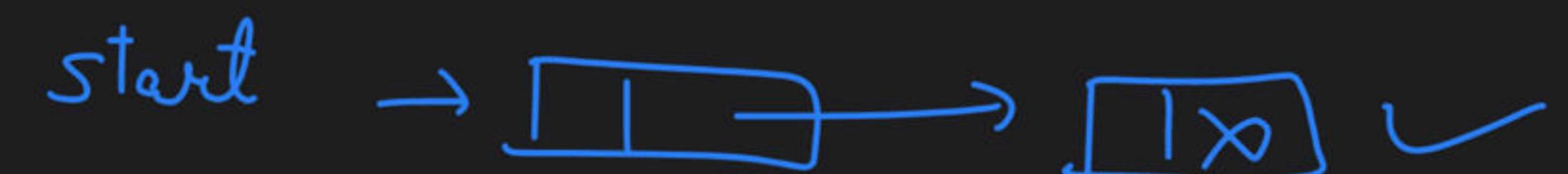
{  
     $p = p \rightarrow link;$

}

return  $p;$

$\Theta(n)$

Valid NULL Terminated linked-list :-

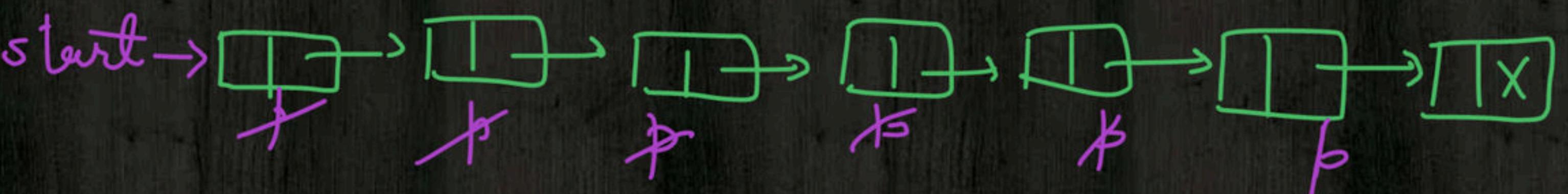


$\text{start} = \text{NULL}$  ✓

# Question

What would be the output after the following code segment is executed on a valid NULL terminated singly linked list.

```
struct node *p;  
p = start;  
while(p → link → link)  
{  
    p = p→link;  
}  
printf("%c", p→data);
```



`start = NULL`  
`start → [ ] X`

Ans:- either NPD or  
data of second last node.

if ( $p == \text{NULL}$  ||  $p \rightarrow \text{link} == \text{NULL}$ )      or      if ( $\neg p$  ||  $\neg p \rightarrow \text{link}$ )  
return  
while ( $p \rightarrow \text{link} \neq \text{NULL}$ )

## Insert' in Linked - list

Can be done anywhere according to requirement.

Whenever a new node to be inserted

- ① First create a new node dynamically using malloc()
- ② Insert this new node.

Creating a new node:-

struct node \*n = (struct node \*)malloc(sizeof(struct node))



if (n == NULL)

{  
    no memory left  
}

# Insertion in Linked List

1. Insertion at beginning
2. Insertion after given node
3. Insertion at the end

# Insertion at beginning



Insertion (start, item)

{

n → data = item

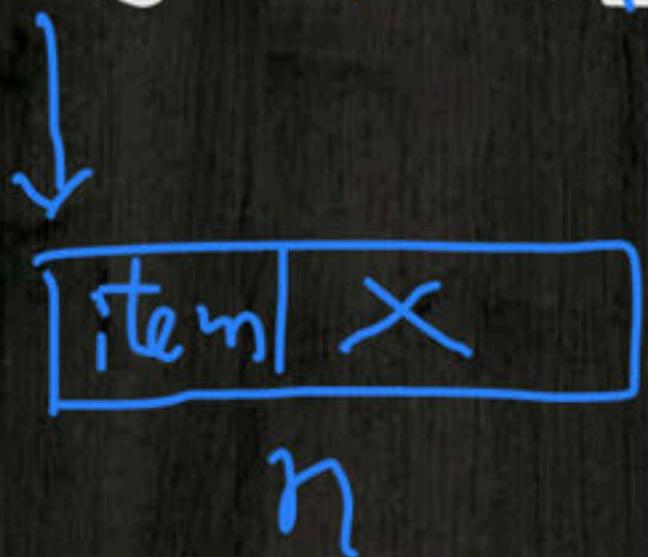
n → link = start

start = n;

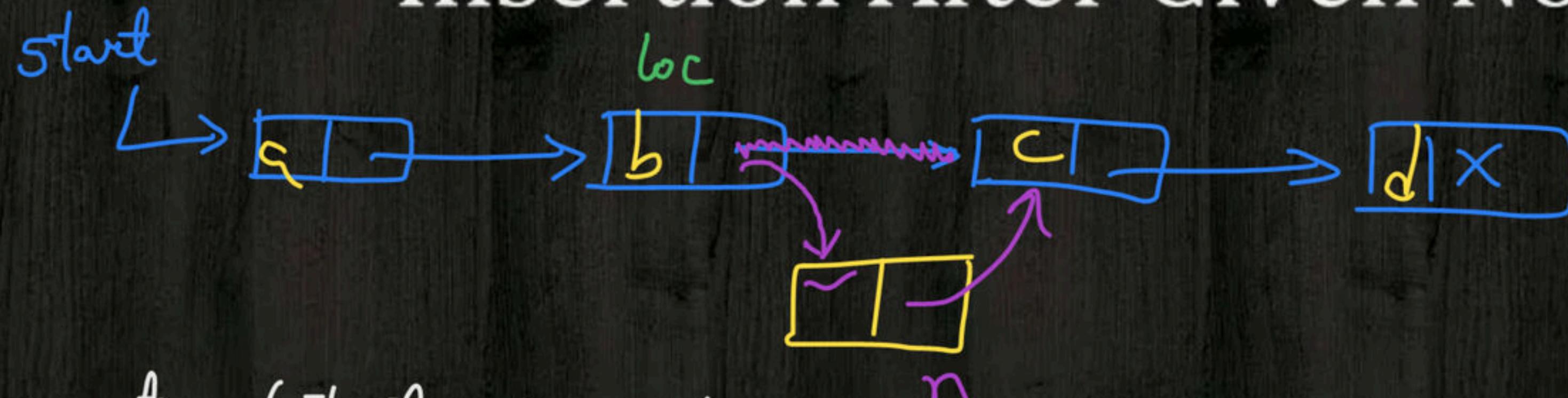
}

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

Start ≠ NULL



# Insertion After Given Node



Insulation (*start*, *loc*, *item*)

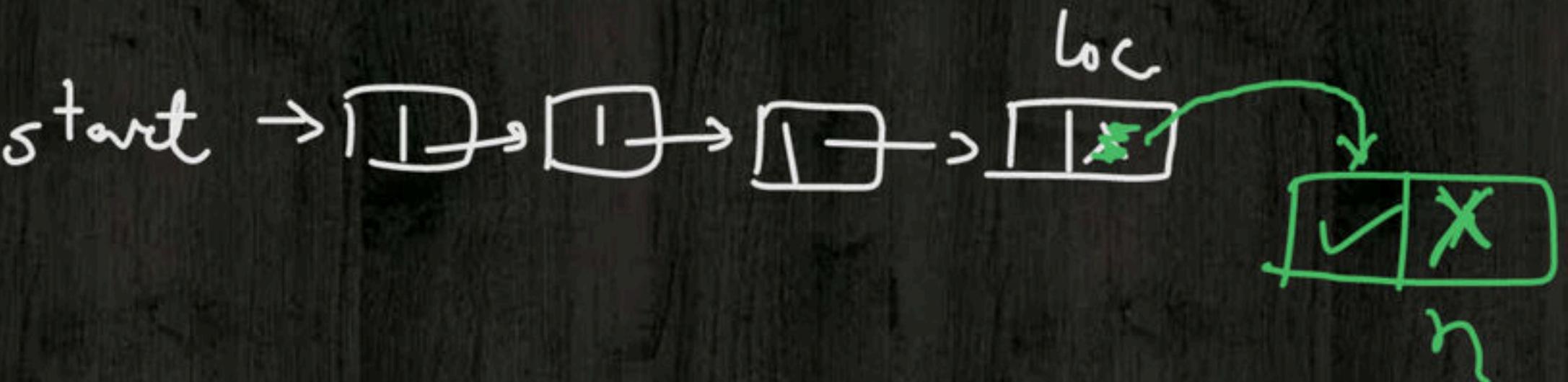
{  
    *n* → *data* = *item*

*n* → *link* = *loc* → *link*

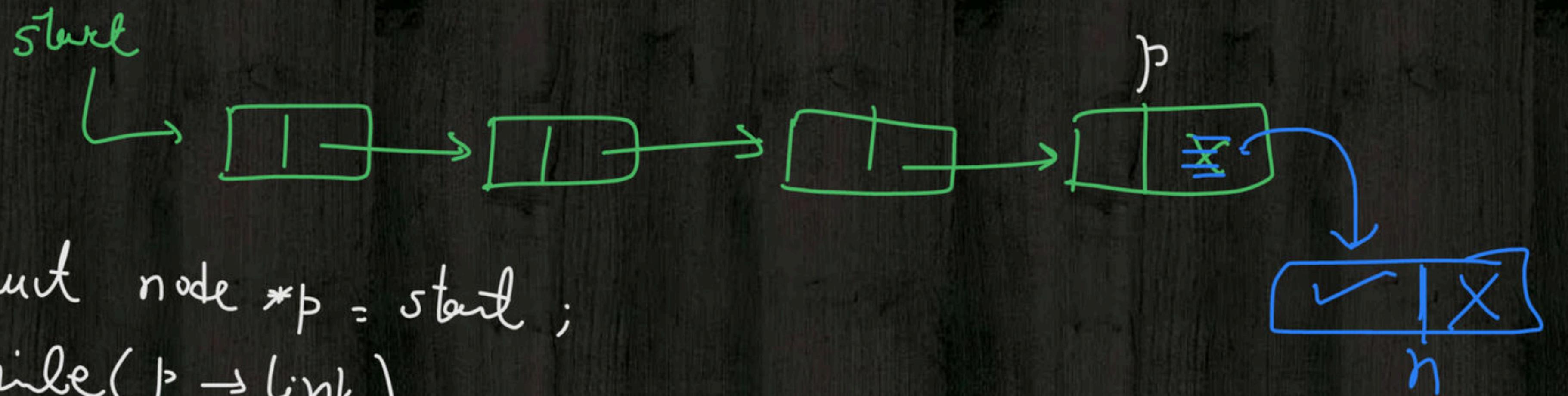
*loc* → *link* = *n*

}

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



# Insertion At the End



```
struct node *p = start ;
```

```
while( p->link )
```

```
{ p = p->link
```

```
}
```

```
h->data = item
```

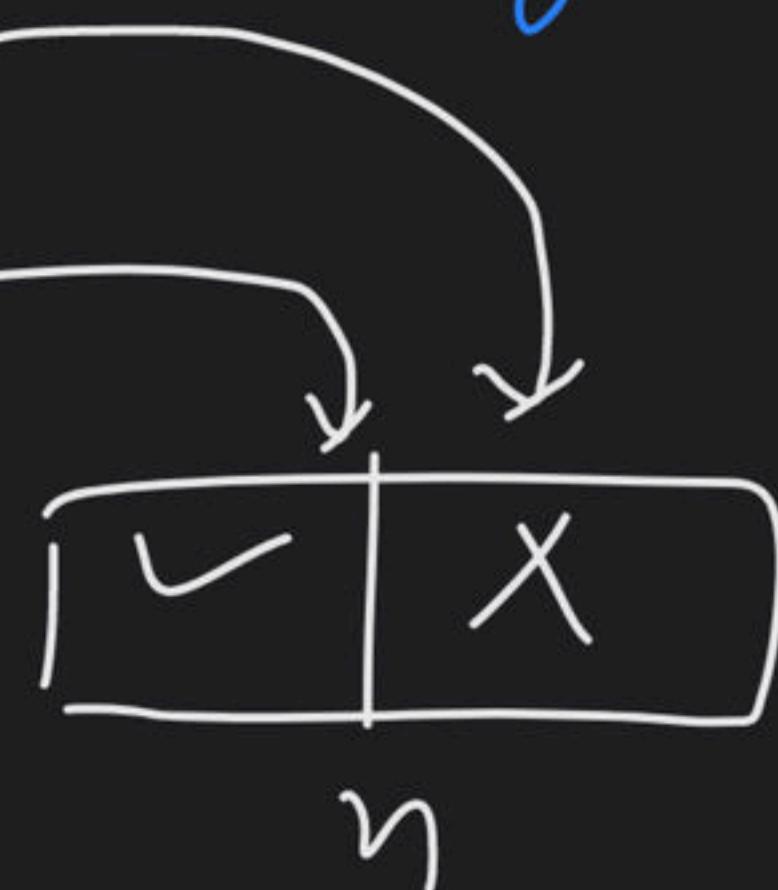
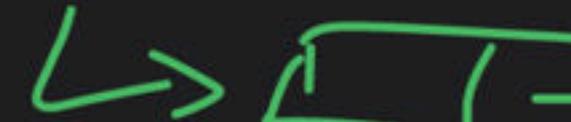
```
n->link = NULL
```

```
p->link = n
```

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

Insert<sup>n</sup> at the end when add. of last node is given

start



$n \rightarrow \text{data} = \text{item}$

$n \rightarrow \text{link} = \text{NULL}$

$\text{last} \rightarrow \text{link} = n$

$\text{last} = n$

$\Theta(1)$

# Deletion in Linked List

1. Deletion at beginning
2. Deletion of given node
3. Deletion at the end

# Deletion At Beginning

# Deletion Of a Given Node

Given a linked list and a key value, delete the node containing the key.

Approach:

- Traverse the list until the node before the target node is found.
- Set the next pointer of the previous node to skip the target node.

Time Complexity: O(n) where n is the number of nodes in the list.

Space Complexity: O(1) as no extra space is required.

Implementation:

```
function deleteNode(head, key) {  
    if (head === null) return null;  
  
    if (head.data === key) {  
        if (head.next === null) return null;  
        else return head.next;  
    }  
  
    let current = head;  
    while (current.next !== null) {  
        if (current.next.data === key) {  
            current.next = current.next.next;  
            break;  
        }  
        current = current.next;  
    }  
  
    return head;  
}
```

Conclusion:

The function `deleteNode` takes the head of a singly linked list and a key value as input. It returns the modified list where the node containing the key has been deleted. If the list is empty or the key is not found, it returns the original list.

The time complexity of this approach is O(n), where n is the number of nodes in the list. This is because we need to traverse the list to find the node before the target node. The space complexity is O(1) as we are not using any additional data structures.

The implementation uses a while loop to traverse the list. It starts at the head and moves to the next node until it finds the node before the target node. Once it finds the correct node, it sets its next pointer to skip the target node. Finally, it returns the modified list.

This approach is efficient for singly linked lists as it only requires a single pass through the list. However, it may not be the most efficient for doubly linked lists or other more complex data structures.

# Deletion At the End

# Question 1 GATE-2004

Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership and cardinality will be the slowest?

- (A) Union only
- (B) Intersection, membership
- (C) Membership, cardinality
- (D) Union, intersection

# Application of Linked List

- ◆ To store polynomials
  - Univariate
  - Bivariate



*DPP*



# Question 1

Write an algorithm to convert the list into a circular list?

## ✓ Question 2

Following C-like function takes a singly-linked list of integers as a parameters and rearranges the elements of list. The function is called with the list containing the integers 3, 5, 5, 5, 7, 8, 9, 9, 9, 9, 12, 15, 18, 19, 23 in the given order. What will be the total no. of contents of the list after the function completes execution (no. of nodes in the list)

```
struct node
{
    int data;
    struct node * next;
};
```

# Question 2

```
void rearrange (struct node * head)
{
    struct node * current = head;
    if( current == NULL) return;
    while (current → next != NULL)
    {
        if (current → data == current → next → data)
        {
            struct node *P = current → next → next;
            free (current → next);
            current → next = p;
        }
        else
        {
            current = current → next;
        }
    }
}
```

# ✓ Question 3

Consider the following function on a valid NULL terminated list:

```
int func(node *start)
{
    struct node *p;
    p = start;
    while(p->link)
    {
        p = p->link;
    }
    return 1;
}
```

The above function returns:

- (A) 1 always
- (B) None always
- (C) Will cause error always
- (D) Error or returns 1

# \ Question 4

**There is node pointer x, which points to a node of a singly linked list. Delete the node after the node pointed by pointer x.**

# Question 5

**There is node pointer x, which points to a node of a singly linked list. Delete the node before the node pointed by pointer x.**

# Question 6

What would be the output after the sequence of steps:



*struct node \*p;*

*p = start → link → link;*

*start → link → link → link = p → link → link;*

*p → link → link → link = start;*

*printf("%c", start → link → link → link → link → data);*

# Question 7

What would be the output after the sequence of steps:



```
struct node *p, *q;
```

```
p = start → link → link;
```

```
q = p → link → link;
```

```
p → link = q → link;
```

```
q → link → link = start;
```

```
printf("%c", start → link → link → link → data);
```

# Question 8

What would be the output after the sequence of steps:



```
struct node *p, *q;
```

```
p = start → link;
```

```
q = p → link → link → link;
```

```
p → link = q → link → link;
```

```
q → link → link = p;
```

```
printf("%c", start → link → link → link → data);
```

---

# Happy Learning



---