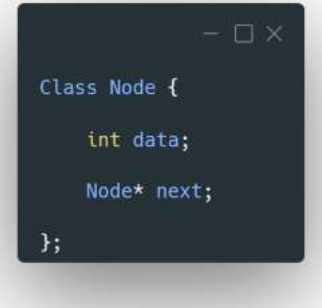
3/11/2023

LINKED LIST CLASS - 1

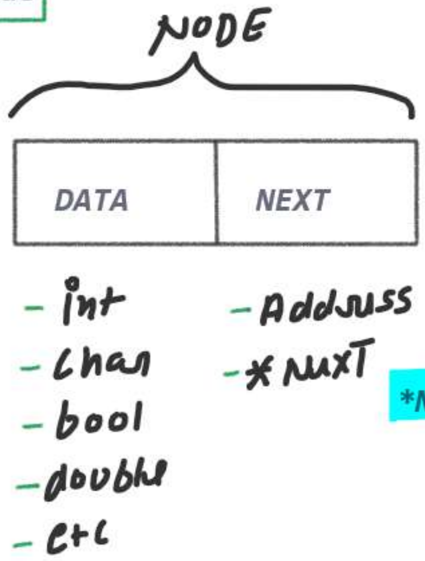
1. What is a node?

1. Nodes make up linked list
2. Each node is composed of **data** and a **reference to the next node** in the sequence.
3. **Last node has always a reference to null which indicates the end of the linked list.**
4. Head node is starting node and Tail node is ending node of linked list.
5. **Head and tail will have a null reference when linked list is empty.**

Visualisation of a node



```
Class Node {
    int data;
    Node* next;
};
```



int *PTR;

*PTR pointer points to an integer

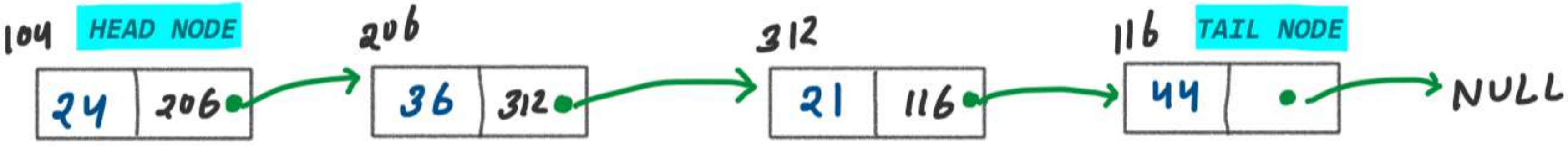
*NEXT pointer points to a Node

2. What is a linked list?

MAGICAL LINE BY LOVE BABBAR BHAIYA -->
LINKED LIST IS HINDI

1. It is a linear data structure
2. It is a collection of nodes
3. **It is a sequence of non-continuous memory allocation**
4. Linked list does not follow indexing to access the data
5. **Linked lists use pointers (or references) to access the next node in the sequence, not direct physical memory addresses.**
6. **At runtime/Dynamically, We can shrink and grow size of linked list**

Visualisation of linked list

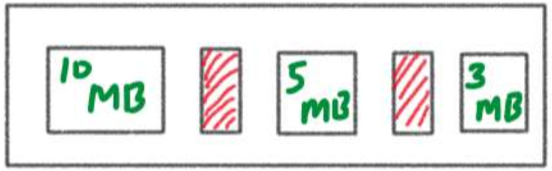


3. Why use of linked list?

1. Efficient use of memory:
Linked lists wastes memory less than array and vector.

Continuous Memory Allocation

Array and vector



Total Free Space = 10 + 5 + 3 = 18 MB

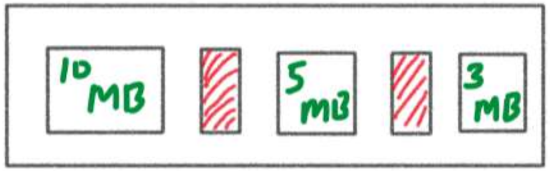
① 8 MB ✓
② 10 MB ✓
③ 11 MB X

MEMORY WASTE

2. Dynamic memory allocation:
Linked lists can be used when the number of elements is not known in advance.

Non-Continuous Memory Allocation

Linked list



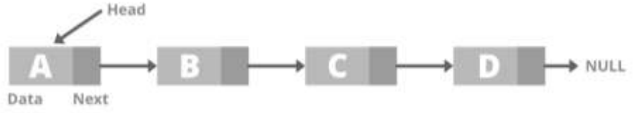
Total Free Space = 10 + 5 + 3 = 18 MB

① 8 MB ✓
② 10 MB ✓
③ 11 MB ✓

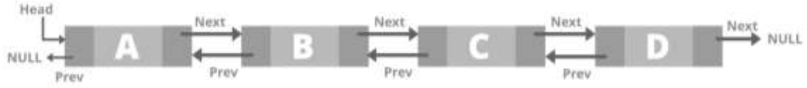
LESS MEMORY WASTE

4. Types of linked list

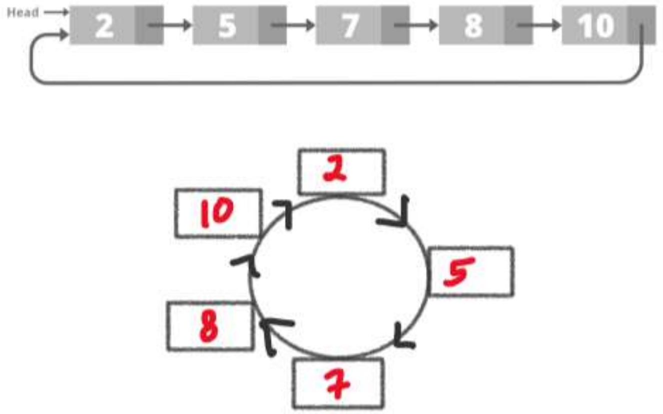
Singly Linked List



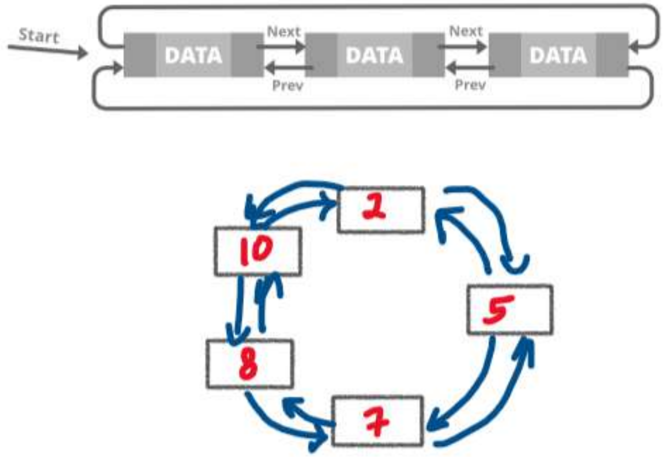
Doubly Linked List



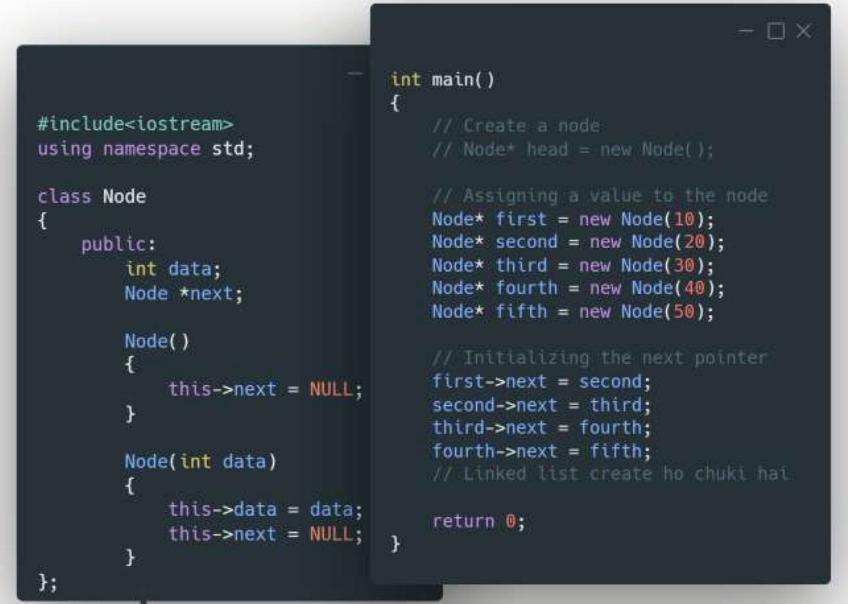
Circular Linked List



Doubly Circular Linked List

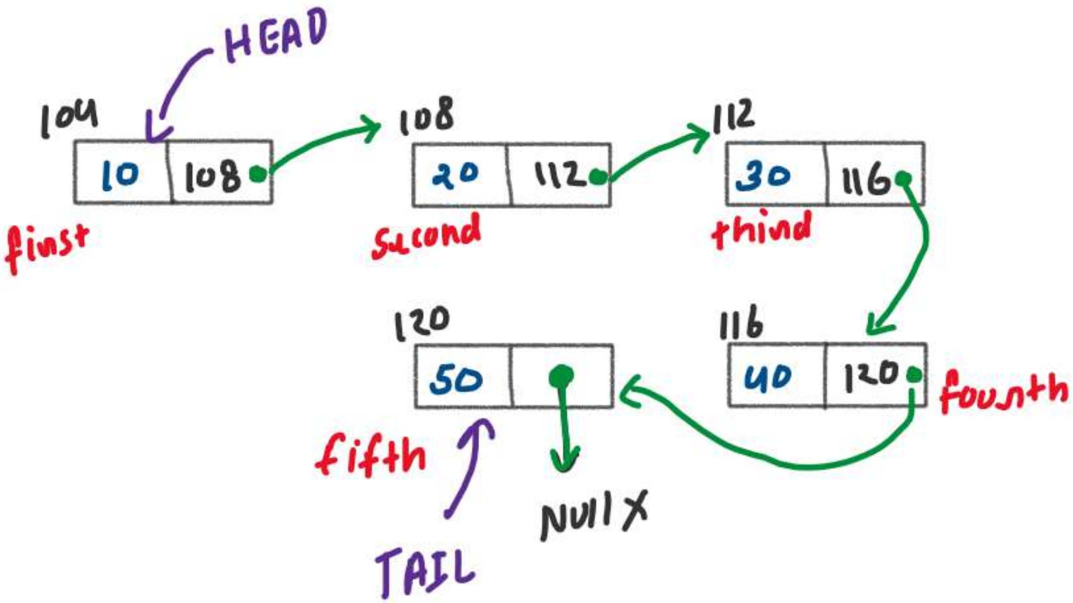


📂 5. Create linked list and a node

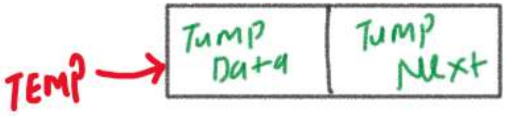


```cpp
#include<iostream>
using namespace std;

class Node
{
    public:
        int data;
        Node *next;

        Node()
        {
            this->next = NULL;
        }

        Node(int data)
        {
            this->data = data;
            this->next = NULL;
        }
};
```

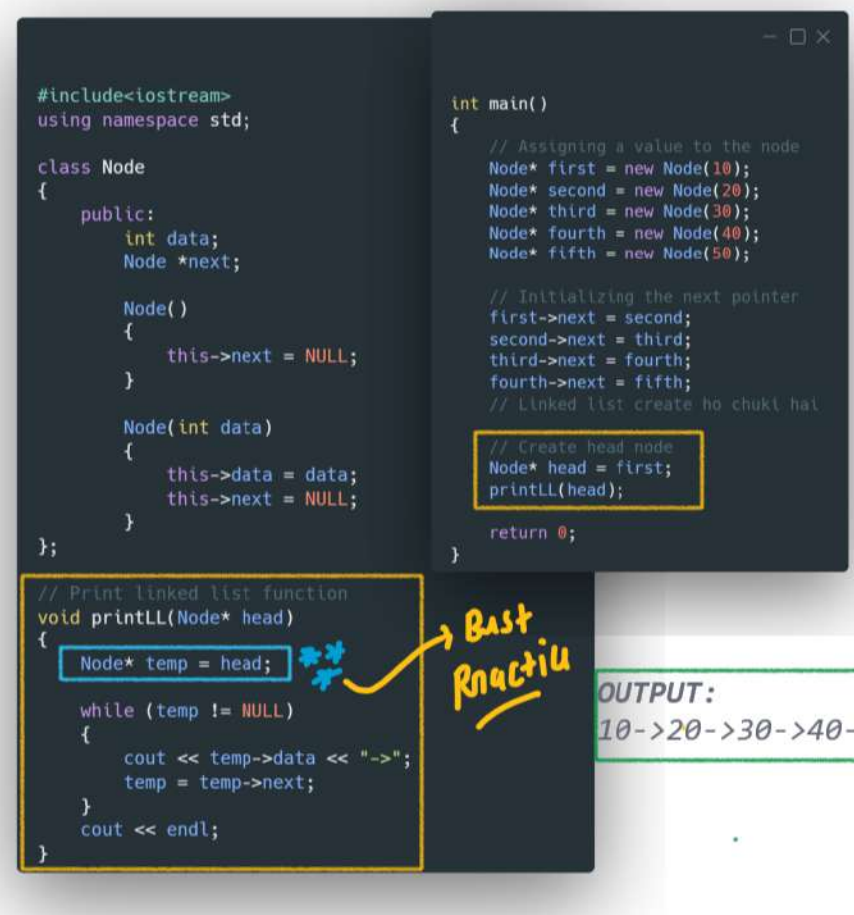
```cpp
int main()
{
    // Create a node
    // Node* head = new Node();

    // Assigning a value to the node
    Node* first = new Node(10);
    Node* second = new Node(20);
    Node* third = new Node(30);
    Node* fourth = new Node(40);
    Node* fifth = new Node(50);

    // Initializing the next pointer
    first->next = second;
    second->next = third;
    third->next = fourth;
    fourth->next = fifth;
    // Linked list create ho chuki hai

    return 0;
}
```



Node* temp = new Node();



6. Print linked list



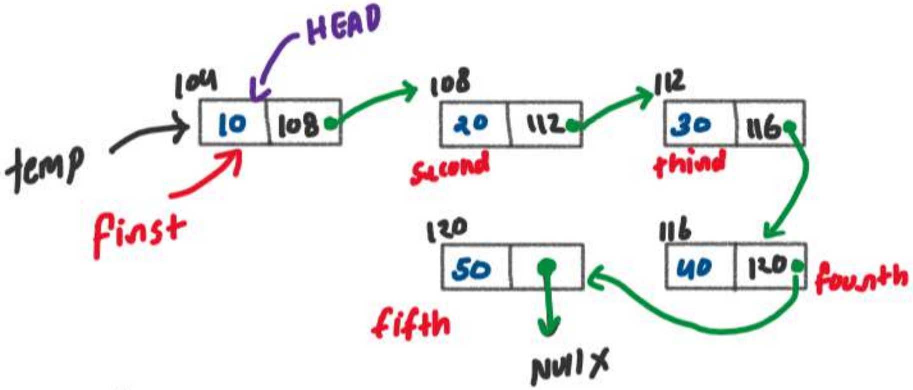
```
#include<iostream>
using namespace std;

class Node
{
    public:
        int data;
        Node *next;

        Node()
        {
            this->next = NULL;
        }

        Node(int data)
        {
            this->data = data;
            this->next = NULL;
        }
};

// Print linked list function
void printLL(Node* head)
{
    Node* temp = head;

    while (temp != NULL)
    {
        cout << temp->data << "->";
        temp = temp->next;
    }
    cout << endl;
}
```

```
int main()
{
    // Assigning a value to the node
    Node* first = new Node(10);
    Node* second = new Node(20);
    Node* third = new Node(30);
    Node* fourth = new Node(40);
    Node* fifth = new Node(50);

    // Initializing the next pointer
    first->next = second;
    second->next = third;
    third->next = fourth;
    fourth->next = fifth;
    // Linked list create ho chuki hai

    // Create head node
    Node* head = first;
    printLL(head);

    return 0;
}
```

** Bast Rnactiu

OUTPUT:

```
10->20->30->40->50->
```



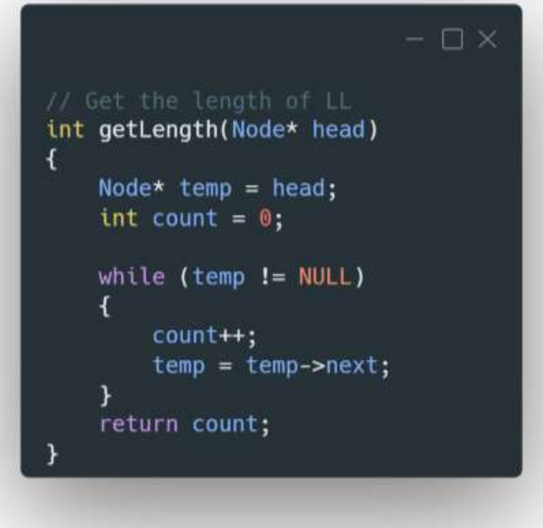
DRY RUN

temp	tump != NULL	Data	tump = tump->next
104	✓	10	108
108	✓	20	112
112	✓	30	116
116	✓	40	120
120	✓	50	NULL
NULL	X END		

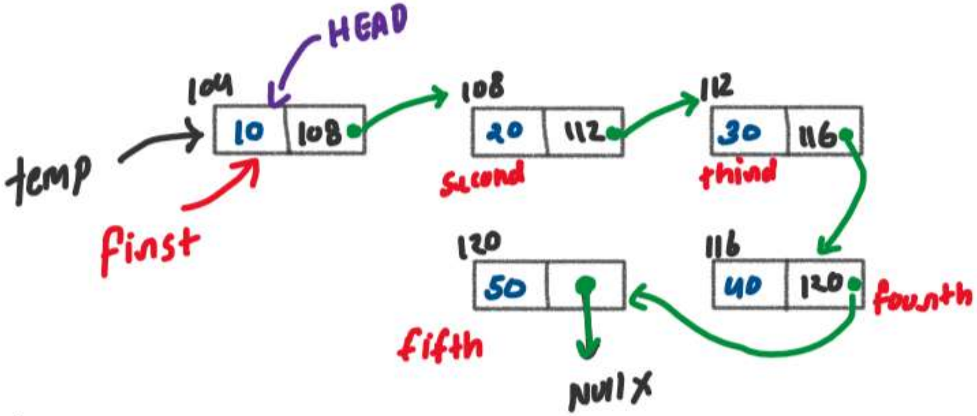
Output

7. Print length of linked list

Print "Number of nodes"



```
// Get the length of LL
int getLength(Node* head)
{
    Node* temp = head;
    int count = 0;

    while (temp != NULL)
    {
        count++;
        temp = temp->next;
    }
    return count;
}
```

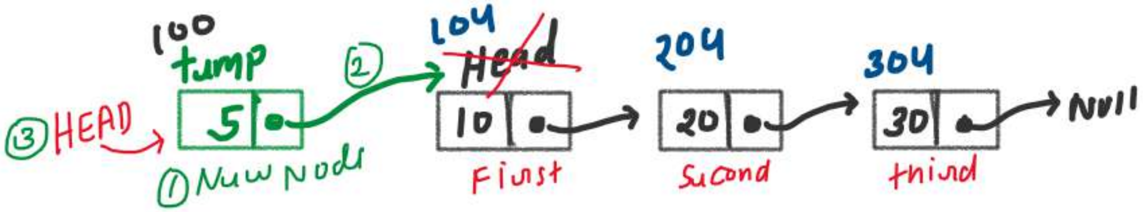


DRY RUN

temp	count	temp!=Null	count++	temp=temp->next
104	0	✓	1	108
108	1	✓	2	112
112	2	✓	3	116
116	3	✓	4	120
120	4	✓	5	Null
Null	5 Output	X END		

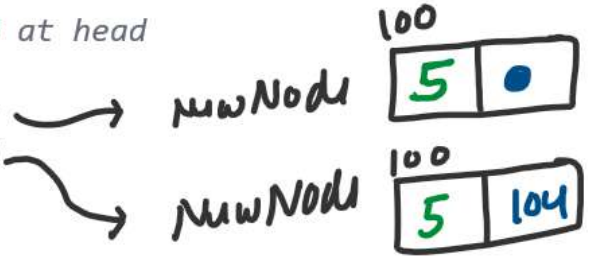
8. Insertion Operations

(I) Insert node at the head

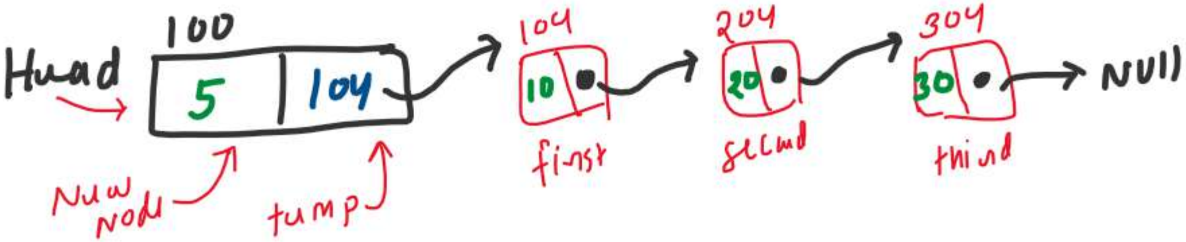


We want to insert **value 5** at head

Step 1: Create a new node
Step 2: temp->next = head
Step 3: head = temp

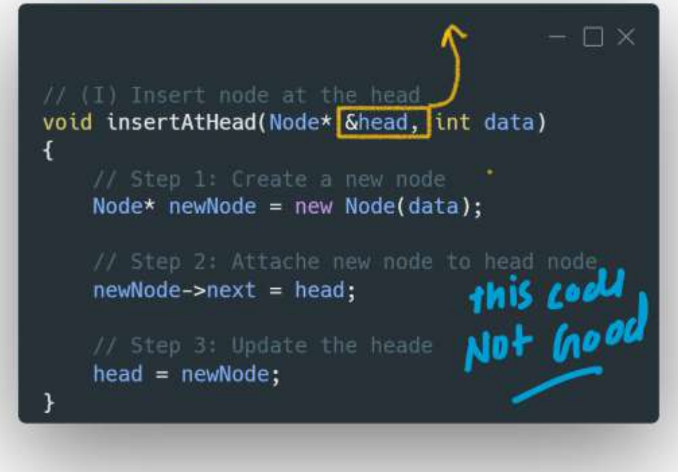


o/p => 5→10→20→30



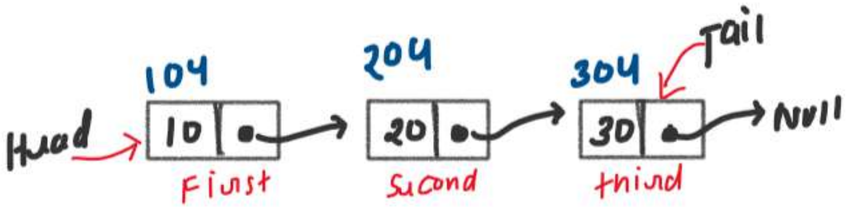
Note

We have to pass head by reference Because can be updated head by reference of new node.



```
// (I) Insert node at the head
void insertAtHead(Node* &head, int data)
{
    // Step 1: Create a new node
    Node* newNode = new Node(data);

    // Step 2: Attache new node to head node
    newNode->next = head;

    // Step 3: Update the heade
    head = newNode;
}
```

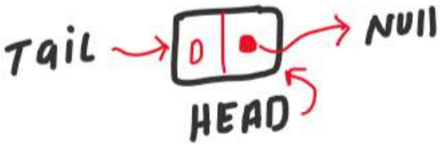
Good Code

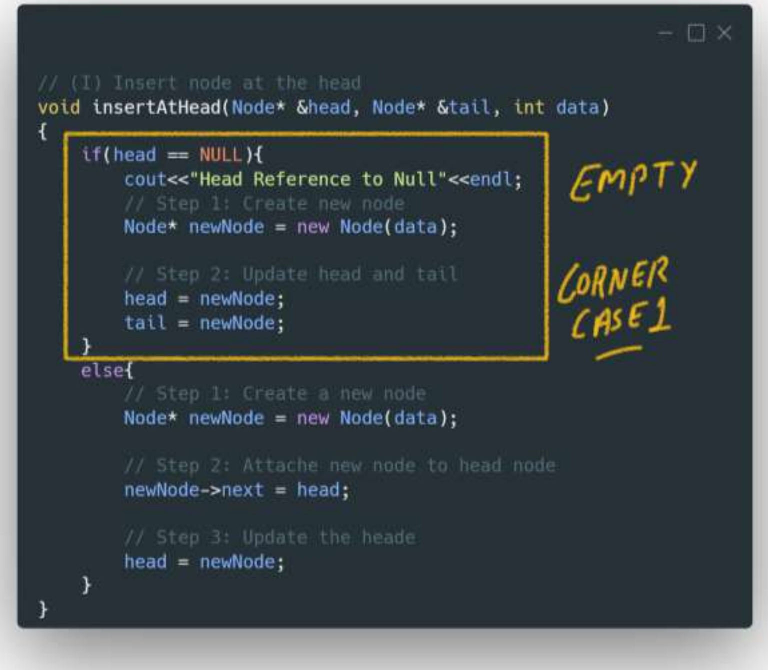


Corner Cases

Corner case 1: **empty linked list**

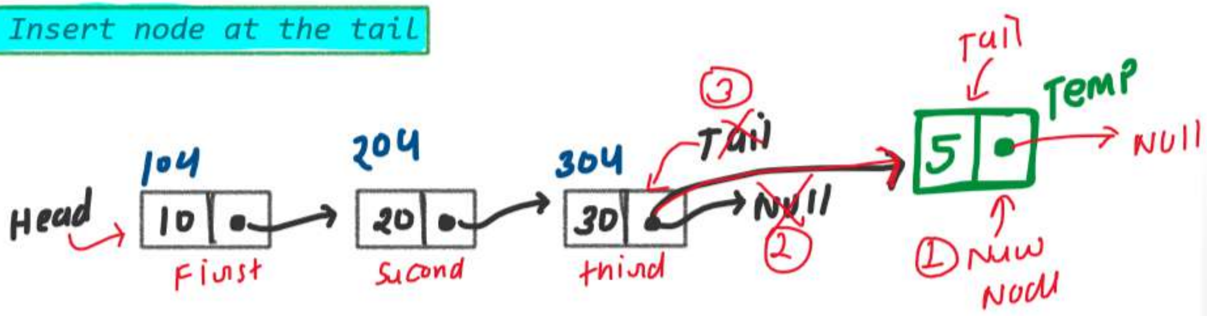
When head and tail reference to null is call empty linked list





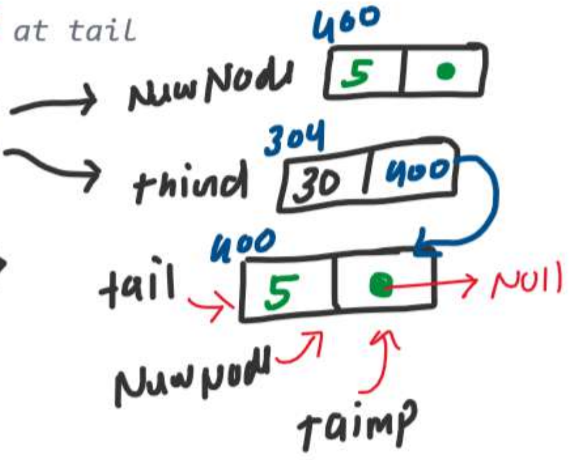
```
// (I) Insert node at the head
void insertAtHead(Node* &head, Node* &tail, int data)
{
    if(head == NULL){
        cout<<"Head Reference to Null"<<endl;
        // Step 1: Create new node
        Node* newNode = new Node(data);

        // Step 2: Update head and tail
        head = newNode;
        tail = newNode;
    }
    else{
        // Step 1: Create a new node
        Node* newNode = new Node(data);

        // Step 2: Attache new node to head node
        newNode->next = head;

        // Step 3: Update the heade
        head = newNode;
    }
}
```

(II) Insert node at the tail



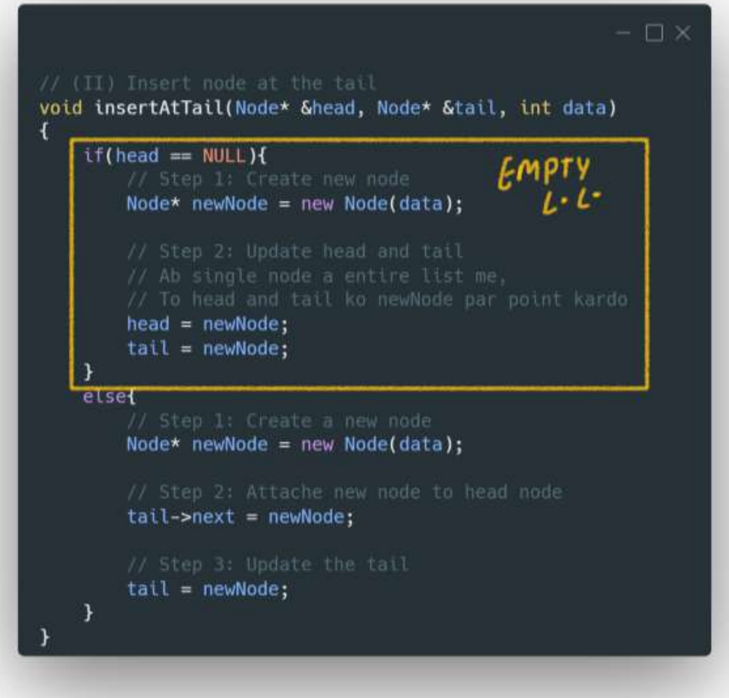
We want to insert **value 5** at tail

Step 1: Create a new node
Step 2: tail->next = temp
Step 3: tail = temp



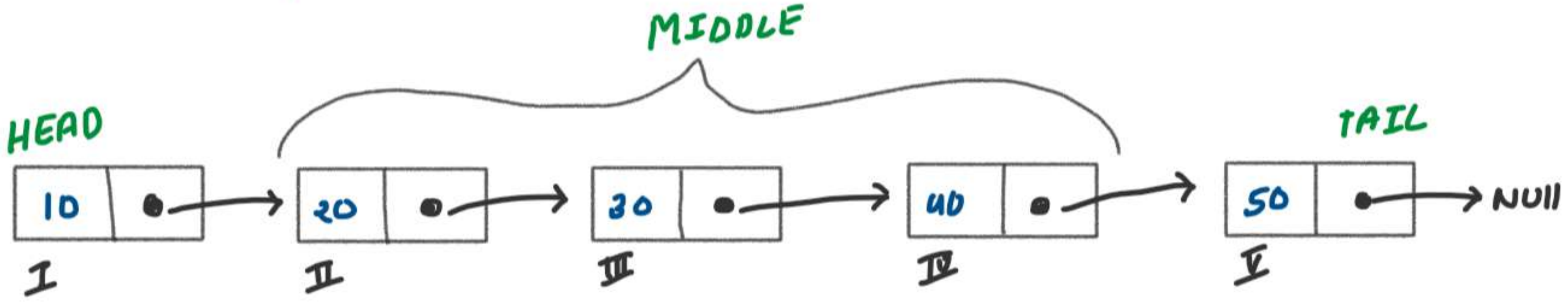
Output

10→20→30→5



```
// (II) Insert node at the tail
void insertAtTail(Node* &head, Node* &tail, int data)
{
    if(head == NULL){
        // Step 1: Create new node
        Node* newNode = new Node(data);

        // Step 2: Update head and tail
        // Ab single node a entire list me,
        // To head and tail ko newNode par point kardo
        head = newNode;
        tail = newNode;
    }
    else{
        // Step 1: Create a new node
        Node* newNode = new Node(data);

        // Step 2: Attache new node to head node
        tail->next = newNode;

        // Step 3: Update the tail
        tail = newNode;
    }
}
```

(III) Insert node at any position



CATCH 1

YAha par GALTI KI THI Bhaiya ne

1) $P < 1$ can't insert

2) $P = 1$ insert at head

3) $P = ~~5+1~~$ $P > 5$ insert at tail

4) $P > 1$ & $P < 5$ insert in middle

5) $P > 5$ can't insert

Length of LL = 5

If we Apply this condition then

lets suppose we want to Insert 500 at position 6

O/P Can't Insert

If we do not apply this condition then

lets suppose we want to Insert 500 at position 51

O/P Runtime ERROR (KYUNKI 51 Koi position Ha Hi Nahi Hai)

↳ segmentation fault

How to Resolve RUN TIME ERROR?

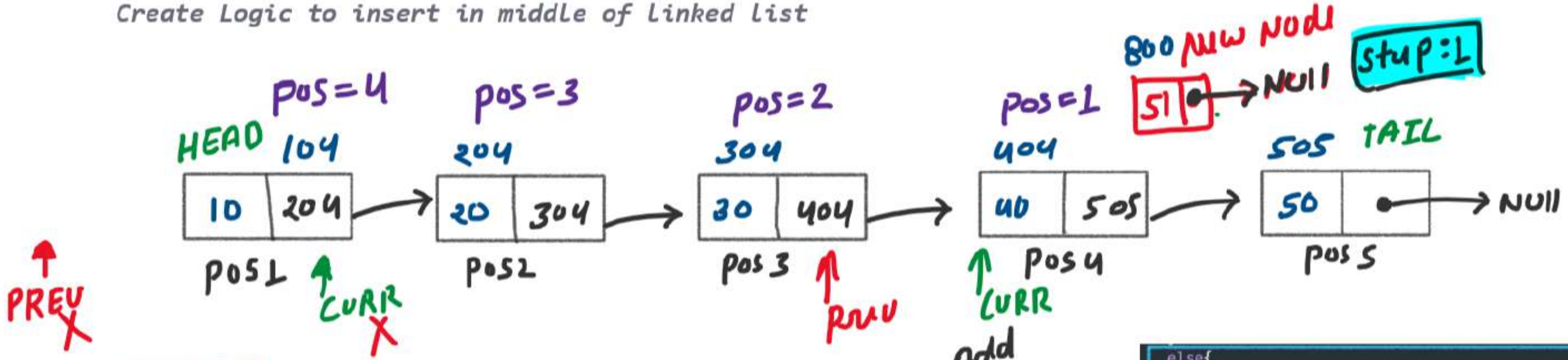
we have to change the ③ condition

③ $P >$ length

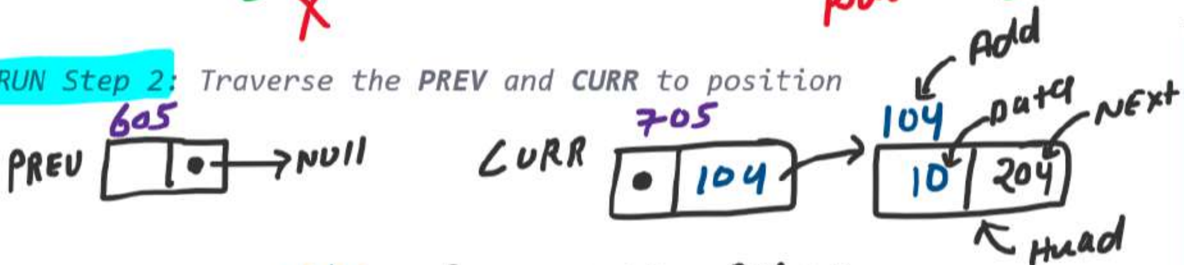
Final condition

1. $P <= 1$ — Insert at Head
2. $P > length$ OR $P > length+1$ — Insert at tail
3. $P > 1$ and $P < length$ — Insert in middle

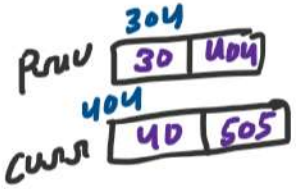
Create Logic to insert in middle of linked list

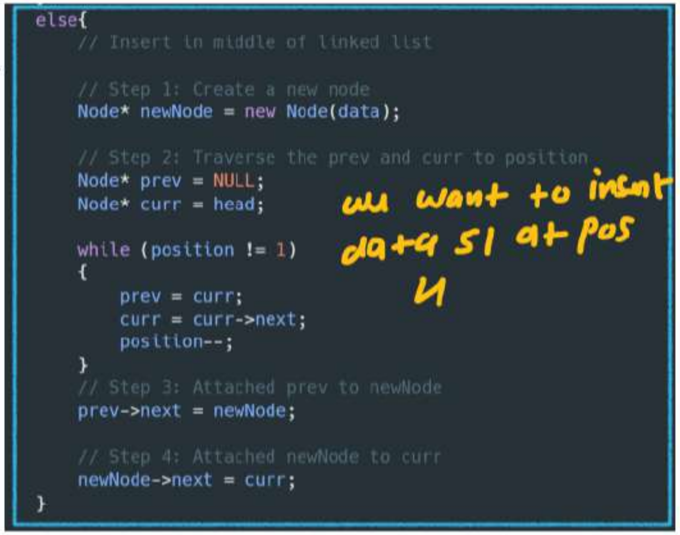


DRY RUN Step 2: Traverse the **PREV** and **CURR** to position



Pos	PREV	CURR	while	Prev	CURR	Pos--
4	Null	104	✓	104	204	3
3	104	204	✓	204	304	2
2	204	304	✓	304	404	1
1	304	404	X END			

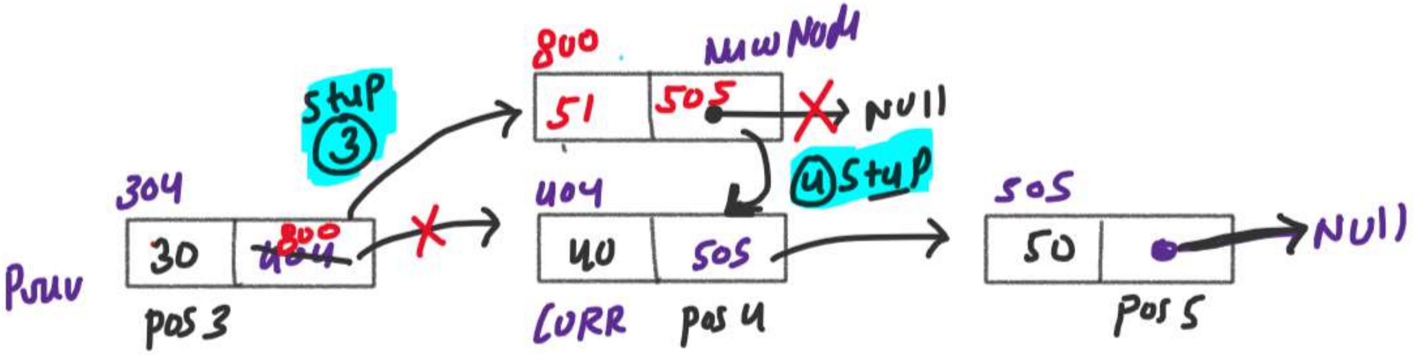




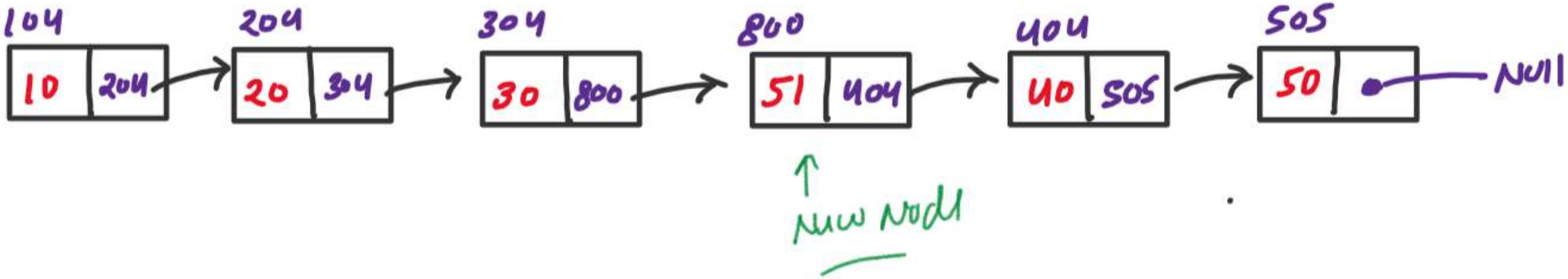
```
else{
    // Insert in middle of linked list

    // Step 1: Create a new node
    Node* newNode = new Node(data);

    // Step 2: Traverse the prev and curr to position
    Node* prev = NULL;
    Node* curr = head;

    while (position != 1)
    {
        prev = curr;
        curr = curr->next;
        position--;
    }
    // Step 3: Attached prev to newNode
    prev->next = newNode;

    // Step 4: Attached newNode to curr
    newNode->next = curr;
}
```

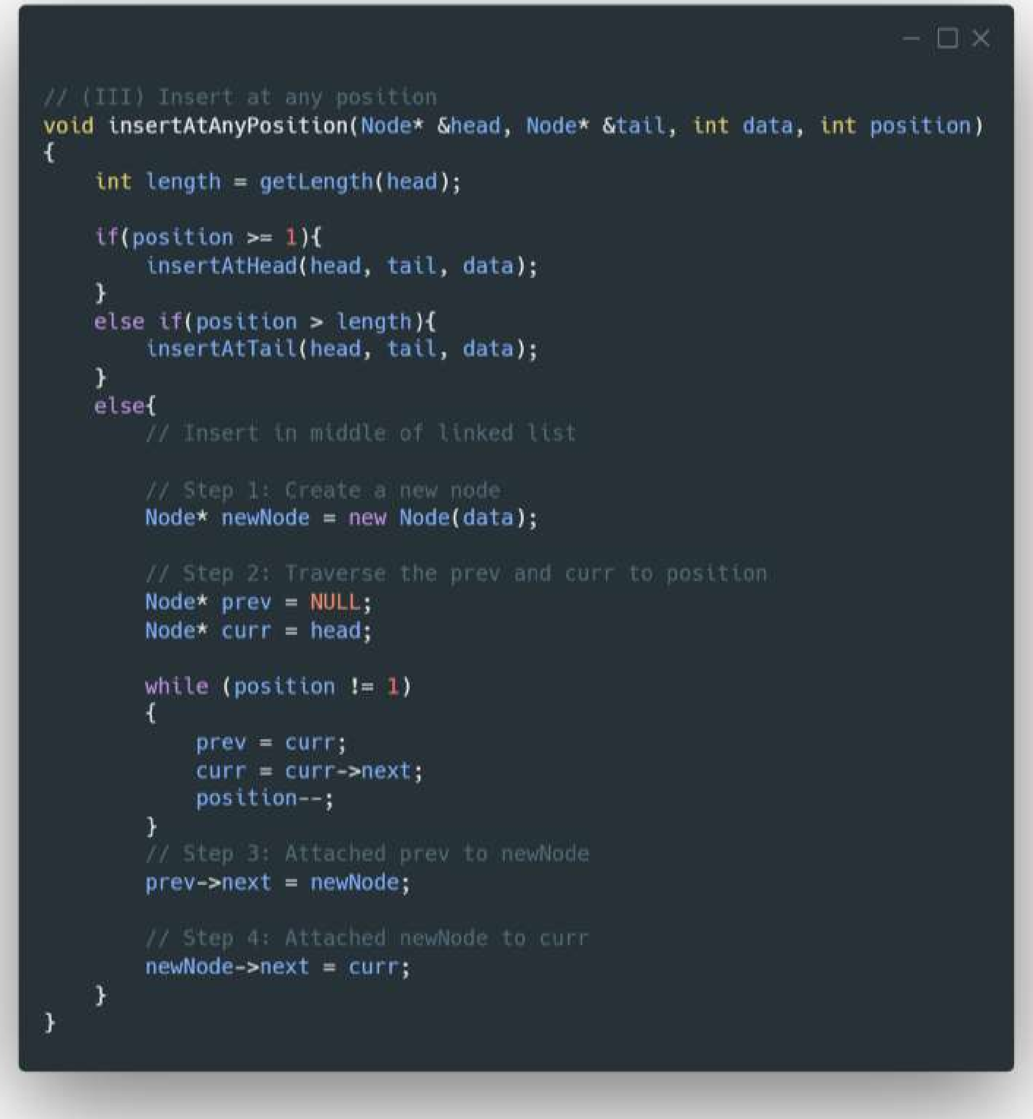
we want to insert data 51 at pos 4

Step 3 and 4



Finally

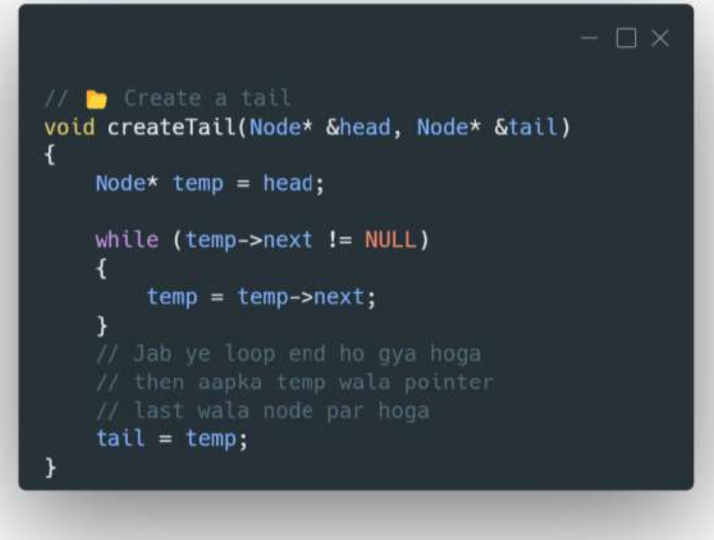




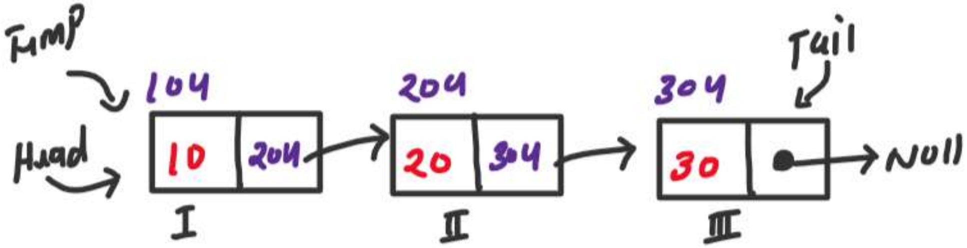
```cpp
// (III) Insert at any position
void insertAtAnyPosition(Node* &head, Node* &tail, int data, int position)
{
    int length = getLength(head);

    if(position >= 1){
        insertAtHead(head, tail, data);
    }
    else if(position > length){
        insertAtTail(head, tail, data);
    }
    else{
        // Insert in middle of linked list

        // Step 1: Create a new node
        Node* newNode = new Node(data);

        // Step 2: Traverse the prev and curr to position
        Node* prev = NULL;
        Node* curr = head;

        while (position != 1)
        {
            prev = curr;
            curr = curr->next;
            position--;
        }
        // Step 3: Attached prev to newNode
        prev->next = newNode;

        // Step 4: Attached newNode to curr
        newNode->next = curr;
    }
}
```

T.C. ⇒ $O(1)$

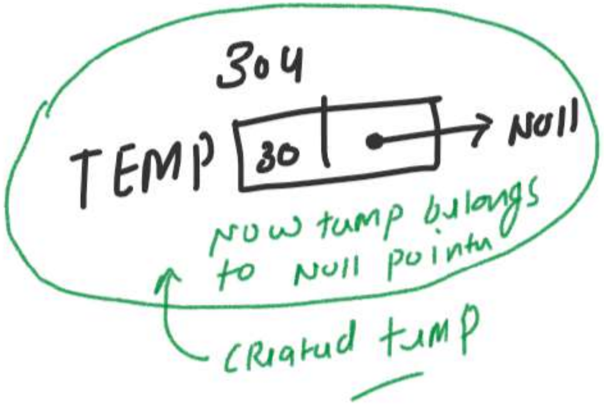
📂 9. Create a tail



```
// 📁 Create a tail
void createTail(Node* &head, Node* &tail)
{
    Node* temp = head;

    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    // Jab ye loop end ho gya hoga
    // then aapka temp wala pointer
    // last wala node par hoga
    tail = temp;
}
```



temp	while	temp
104	204 != NULL ✓	204
204	304 != NULL ✓	304
304	NULL != NULL ✗	END



304
TEMP
30
NULL
Now temp belongs to NULL pointer
Created temp