LINKED LIST

Lecture - 1

A data structure that works on non-continuous memory procedure. That is how it is different from array and vectors.

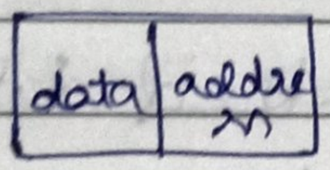
In linked we don't have wastage of memory.

→ In linked list we called it nodes.

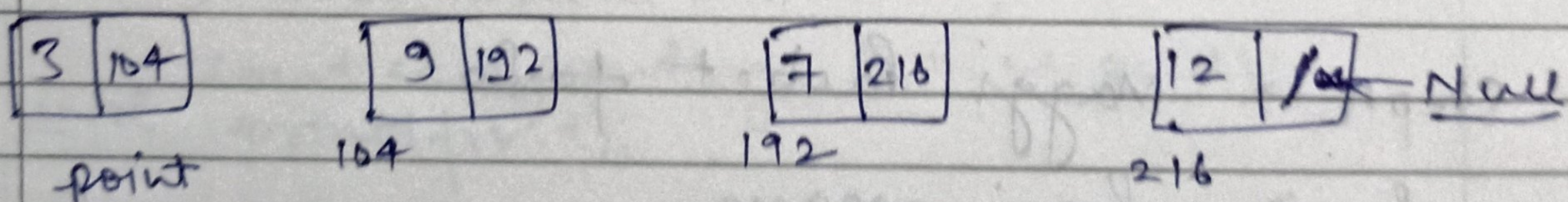
→ In run time we can do whatever we want like performing insertion, deletion or many more.

Linked list is a collection of node.

NODE



→ This is address of next data.



pointer to integer → int *
pointer to NODE → Node *

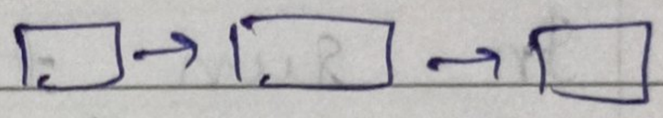
PAGE No
DATE

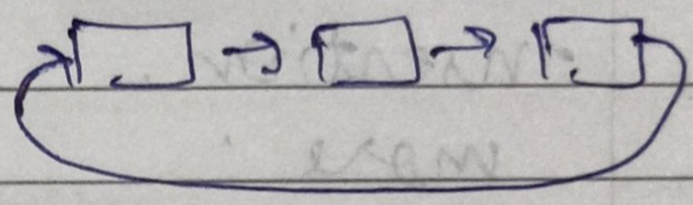
NODE CREATION

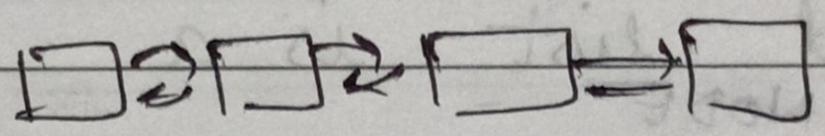
```
class Node
{
    int data
    Node *next;
};
```

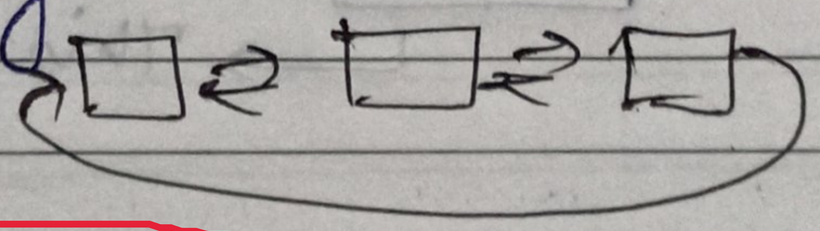
→ Singly linked list

Types of Linked List

1) Singular Linked List 

2) Circular Linked List 

3) Doubly Linked List 

4) Circular Doubly Linked List 

StepA → Maggi Packet daalo.
B → Paani garam
C → Maggi Paani me daalo
D → Masala Daal do.
E → 3 min wait
F → Plate me daal ke kha lo.

(Linked list is Hindi)

PAGE No
DATE

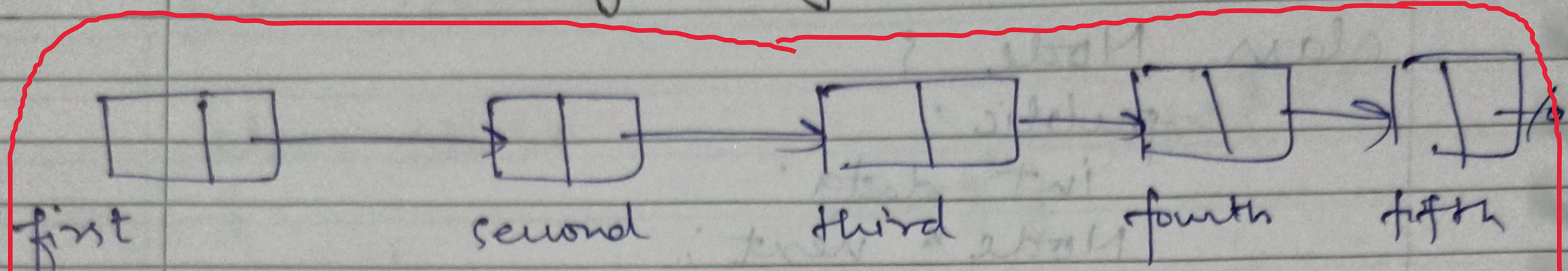
Code

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

Node () {
        this → data = 0;
        this → next = NULL;
    }
Node (int data) {
        this → data = data;
        this → next = NULL;
    }
};
int main() {
        first
    Node * ~~head~~ = new Node (10);
    Node * second = new Node (20);
    Node * third = new Node (30);
                        second;
    first → next = ~~new Node ();~~
    second → next = third;
    cout << " printing LL " << endl;
void print(first);
```

```
void print (Node * head) {
    Node * temp = head;
    while (temp != NULL) {
        cout << temp → data << " ";
        temp = temp → next;
    }
```

PAGE No
DATE

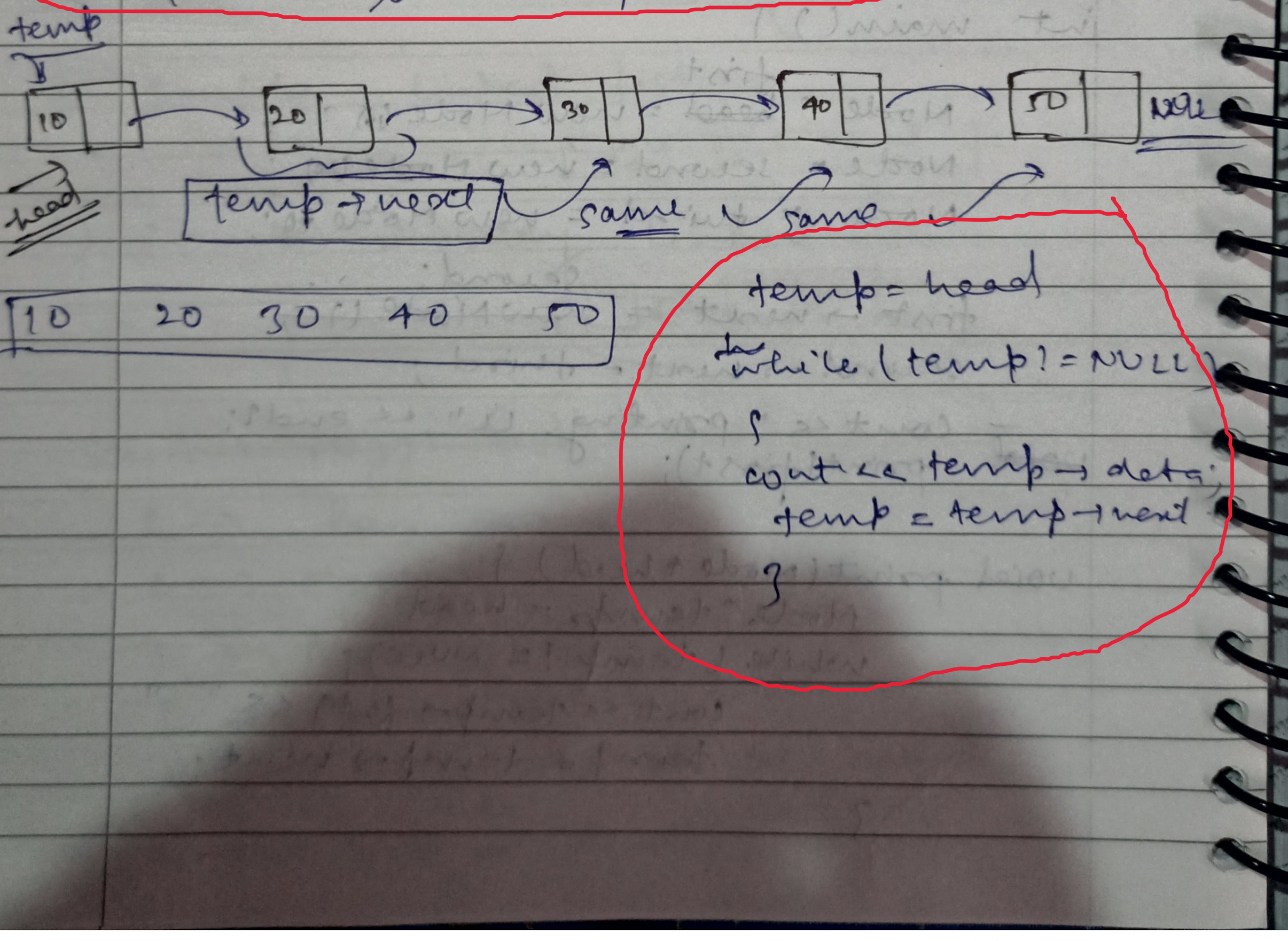
Code mein kya kiya



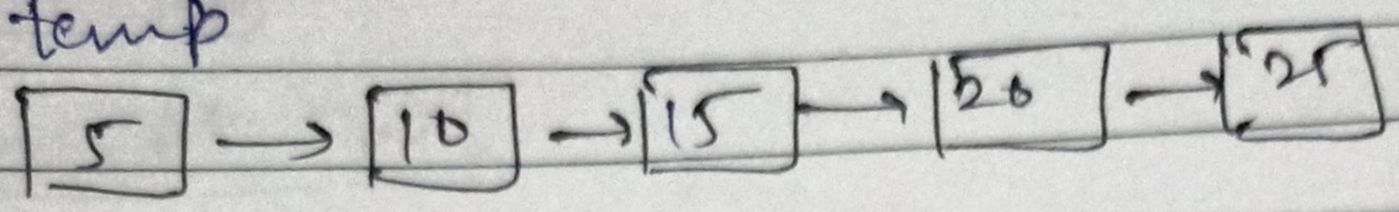
first → next = second
second → next = third
third → next = fourth
fourth → next = fifth.

Print

Print karlo data
Pointer ko aage badao.
Rukna bolda jb null ho.




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



temp → next 10

temp → next → next 15

temp → next → next → next 20

temp → next → next → next → next 25

Insertion

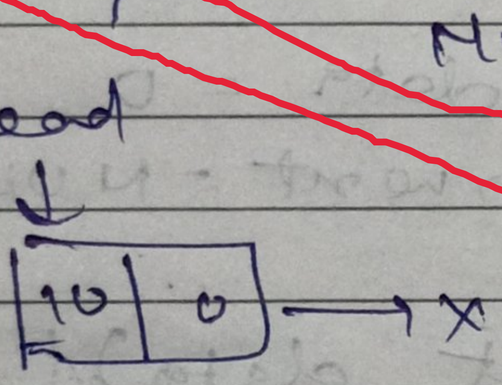
```
class Node {
public:
    int data;
    Node * next;

    Node() {
        this → data = 0
        this → next = NULL;
    }

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

int main()
{
    Node * head = new Node(10)
    insertAtHead (20) ; (head, 20);
    insertAtHead (30) ; (head, 30)
    insertAtHead (60) ; (head, 60)
    insertAtHead (80) ; (head, 90)

    print (head);
    return 0;
}
```

PAGE No
DATE

```
void insertAtHead (Node* & head, int data)
{
    Node* newNode = new Node(data);
    newNode -> next = head;
    head = newNode;
}

void print (Node * head) {
    Node * temp = head
    while (temp != NULL) {
        cout << temp -> data << " ";
        temp = temp -> next;
    }
```

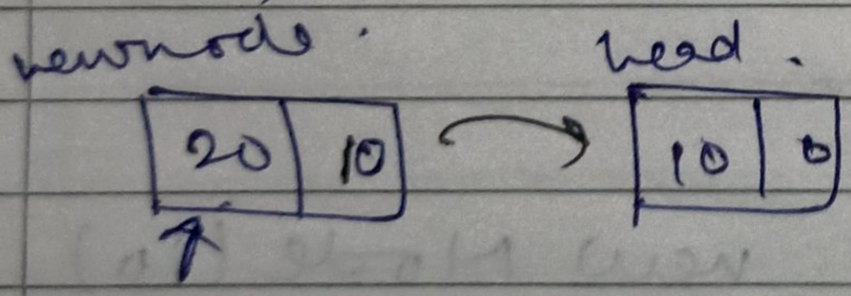
→ Insert

```
Node * head = new Node(10)
```



insert At head (head, 20)

~~1) Create a Node~~



insert At head (head, 20)

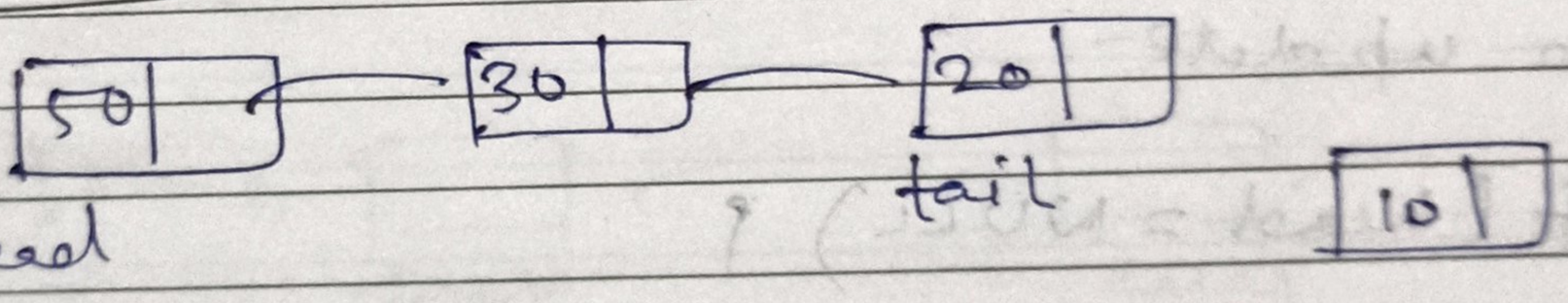
↳ Create a node
↳ newnode → next = head
↳ head = newnode

PAGE No
DATE

Insert At Tail

```
void insertAtTail(Node *tail, int data){

    Node * newNode = new Node(data);
    tail -> next = newNode;
    tail = newNode;

}

int main(){

    insertAtTail(tail, 27);

}
```

How ?.?

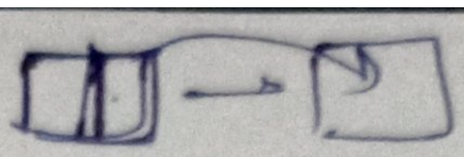


insertAtTail(tail, 10)

→ Create a node

→ tail → next = newNode

→ tail = newNode.



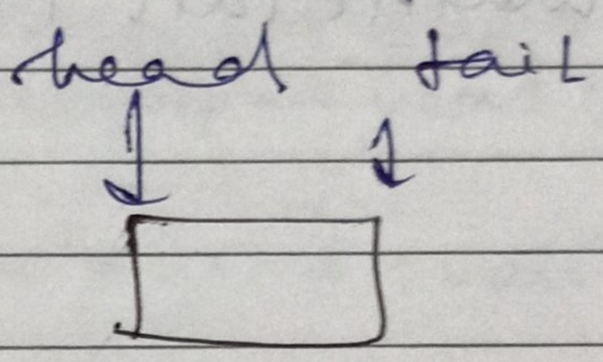
PAGE No
DATE

What if.

{ Head = NULL
  Tail = NULL } → when linked list is empty.

```
if (tail == NULL) {
        tail = newNode
        head = newNode.
}

if (head == NULL) {
        tail = newNode;
}
```

?



head → X

Tail → X

How to update-

```
if (head = NULL) {
        Node * newNode = new Node (data);
        head = newNode;
        tail = newNode;
} return;
```

PAGE No
DATE

Insert at Position

↳ Position ke basis pe

↳ Value ke basis pe :

insertAtPosition (head, tail, data)

↳ check empty

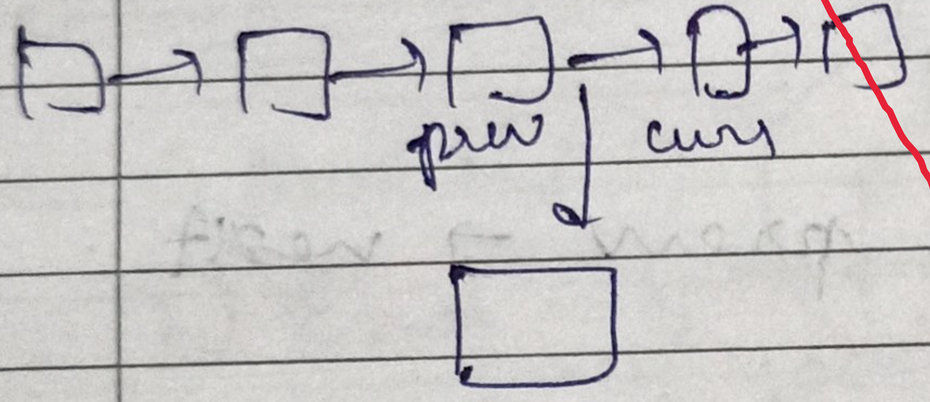
↳ non-empty

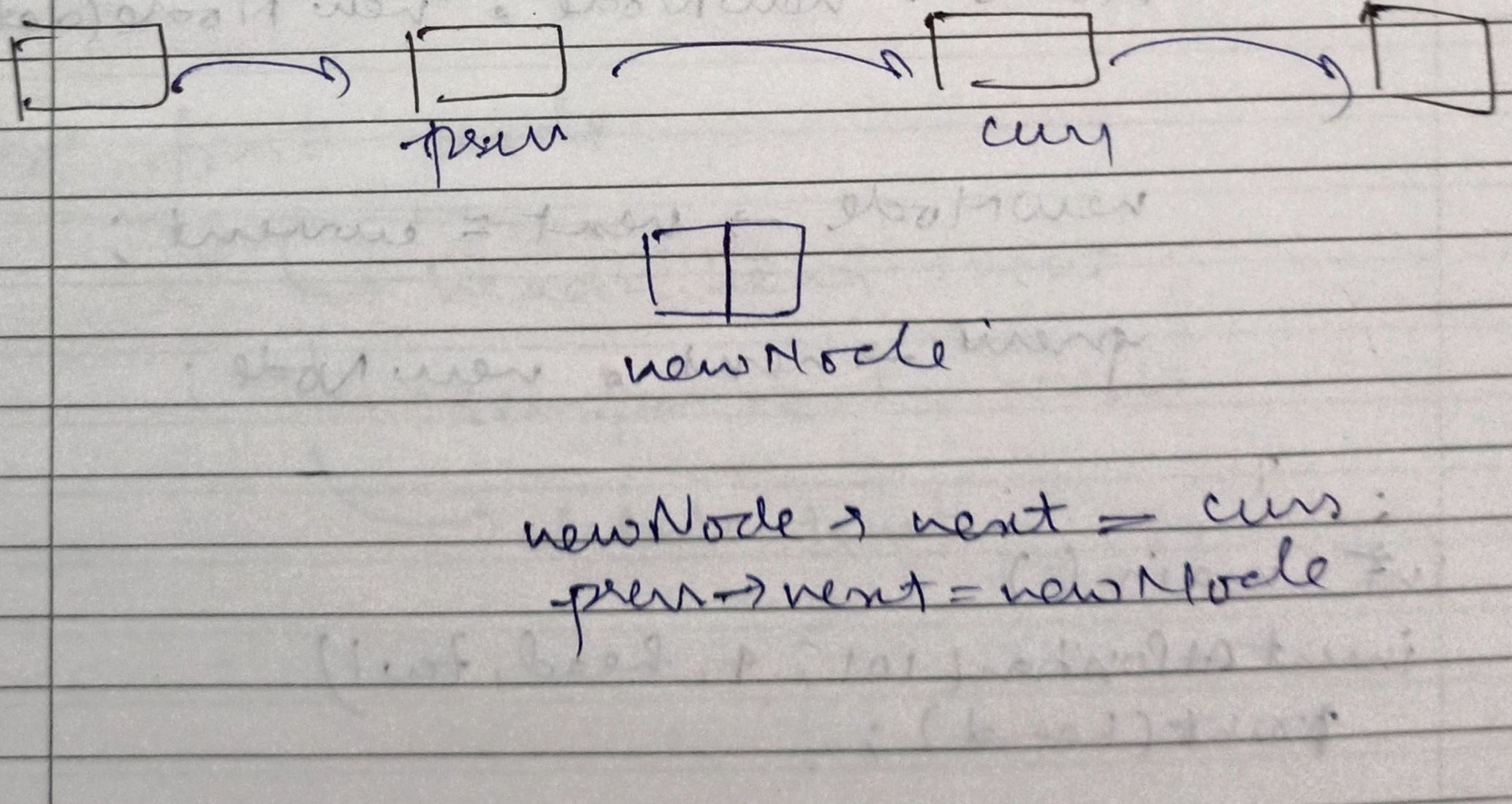
1) find desired position

2) Create a node

3) newNode → next = curr

4) prev → next → newNode





newNode → next = curr ;

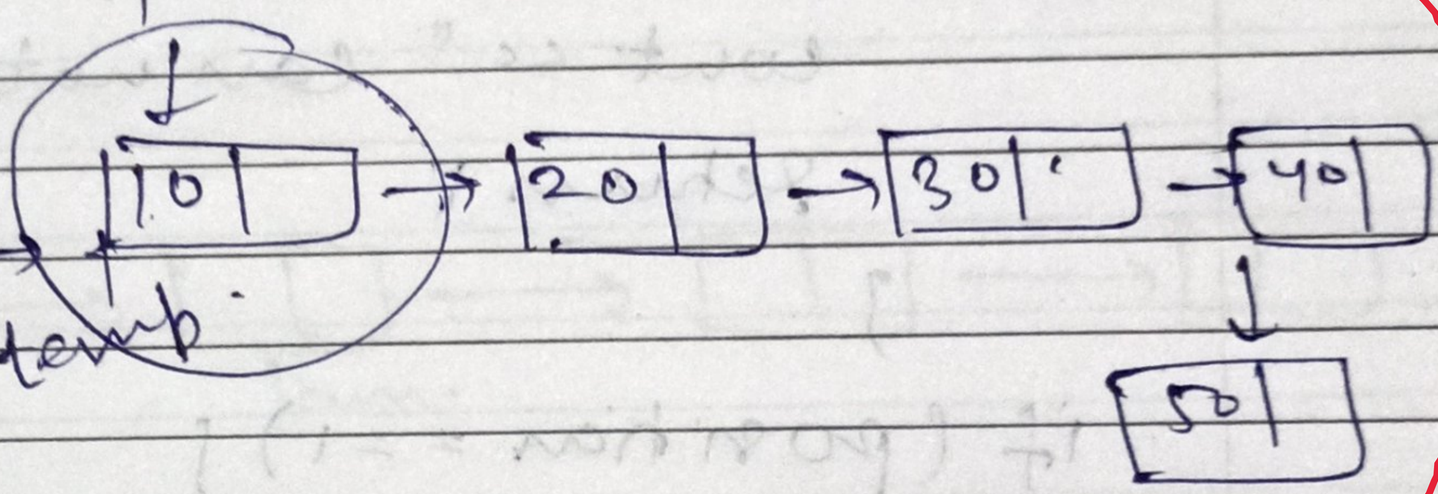
prev → next = newNode ;

```
~~done~~
int len = findLength(head)
if (position >= len)
    { insertAtTail(head, tail, data);
      return;
    }
insertAtPosition( int position, Node * & head
                  , Node * & tail) {
    if (head == NULL) {
        Node * newNode = new Node(data);
        head = NewNode;
        tail = NewNode;
        return;
    }
    if (position == 0) {
        insertAtHead(head, tail, data);
        return;
    }
int i=1;
~~while~~ Node * prev = head;
    while (i < position) {
        prev = prev → next;
        i++;
    }
Node * curr = prev → next;

Node * newNode = new Node(data);

newNode → next = current;

prev → next = newNode;

}
int main()
insertAtPosition(101, 4, head, tail)
    print(head);
```

```
int findlength (Node * & head) {
        Node * temp
        int len=0;
        Node * temp = head;
        while (temp != NULL) {
                temp = temp -> next;
                len++;
        }
        return 0;
}
```

Deletion

1) Middle
2) End.
3) Start

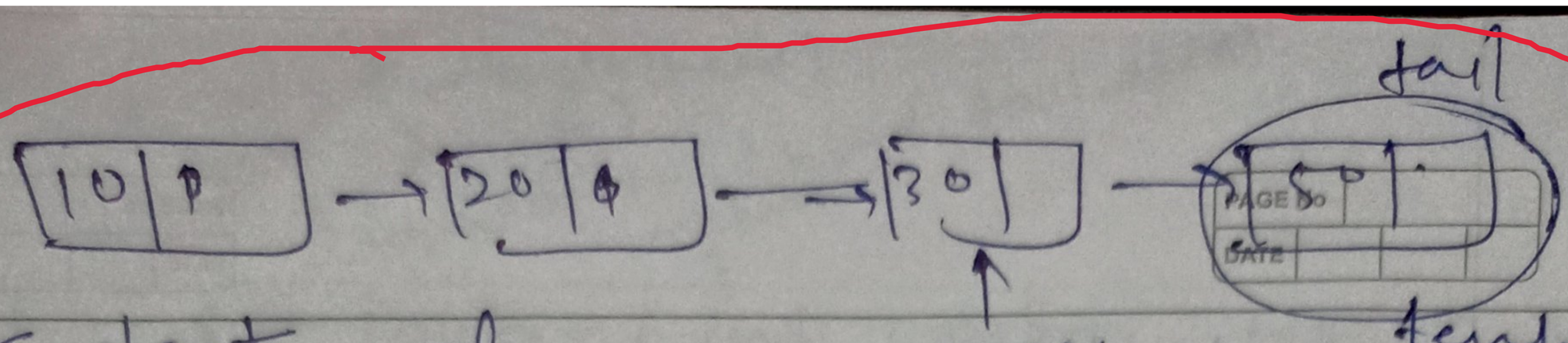


for first node

↳ head = head → next;

↳ temp → next = NULL

↳ delete temp;



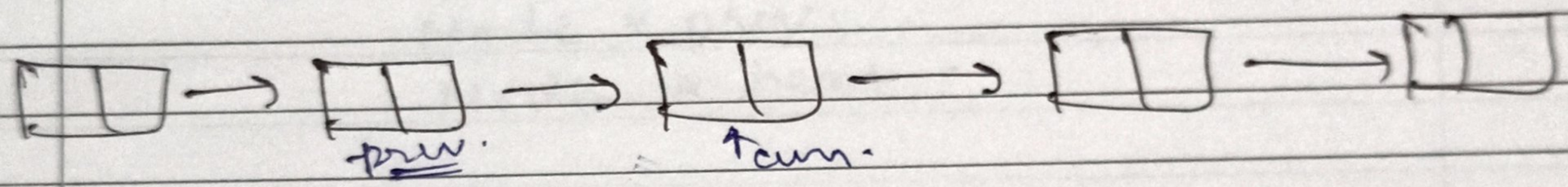
for last node

1) find prev
2) prev → next = NULL
3) tail = prev
4) delete temp

```
void deleteNode (int position, Node * &
                 head, Node * & tail) {

    if (head = NULL) {
        cout << " can not Delete " << endl;
        return;
    }

    if (position == 1) {

        Node * temp = head;
        head = head → next;
        temp → next = NULL;
        delete temp;
        return;
    }

    int len = findLength (head);
    if (position == len) {
        prev → next = NULL;
        Node * temp = tail;
```

```
int i=1;
Node * prev = head;
while (i< position -1 ){
        prev = prev -> next;
        i++;
}
prev -> next = NULL;
Node * temp = tail
tail = prev;
delete temp;
return;
}
}
```

3) Middle



1) find prev
2) prev -> next = curr -> next
3) curr -> next = NULL
4) Delete curr.

Code

```
int i=1
Node * prev = head;
while (i< position -1 ){
        prev = prev -> next;
        i++;
}
Node * current = prev -> next;
```

```
prev -> next = curr -> next ;
curr -> next = NULL ;
delete curr;
```

?