4

Linked list

→ Dynamic Data Structure

one type of linear Data structure, which is made up of collection of Nodes.

NODE :-

Example

ek array [10] ka hai for sum time pe hamne bola ki array ki size 200 karo nhi ho payegi, that is time when linked list comes into picture.

⇒ Linked list ek Dynamic Data Structure hai, ans grows or shrink kar sakte hai during run time and no memory wastage.

⇒ Linked list mai insertion or deletion easy hota hai no shifting is required as such in an array.

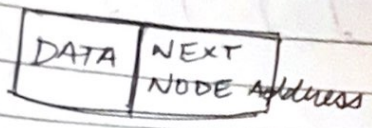
Types of linked list :-

1. Singly linked list
2. Doubly linked list
3. Circular linked list
4. Circular doubly linked list

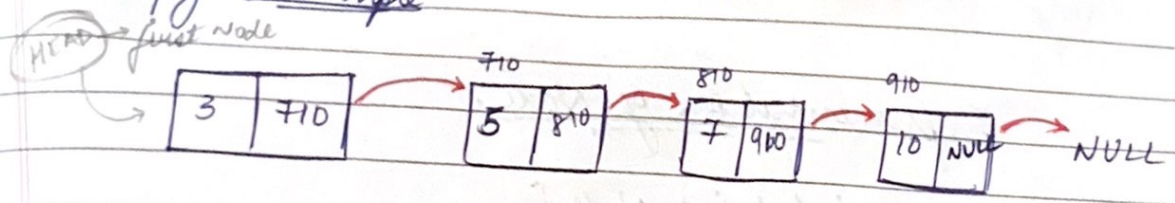
Linked list

One type of linear Data
up of collection of Nod

NODE:-



~~Example~~ Example



⇒ Linked list ek Dynamic Data Structure hai, ans grow or shrink kar sakte hai during run time and no memory wastage.

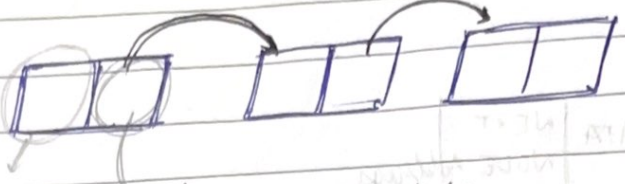
⇒ Linked list mai insertion or deletion easy hota hai no shifting is required as such in an array.

Types of linked list :-

1. Singly linked list
2. Doubly linked list
3. Circular linked list
4. Circular doubly linked list

Singly Linked List

Linked List ⇒ collection of Nodes.
↳ Data and Address



Data of integer type

pointer of Node type

Implementation of Node :-

```
class LinkedListNode {
    public:
        int data;
        LinkedListNode * Next;
};
```

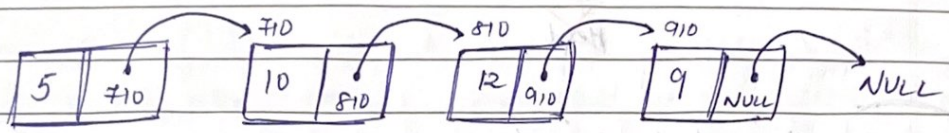
Code :-

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

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

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

par abhi linked list mai head pointer toh hoga hi which indicates start of linked list

```
int main(){
    Node* node1 = new Node(10);
    cout << node1 -> data << endl; → 10
    cout << node1 -> next << endl; → 0x0
    return 0;
}
```

Singly linked list



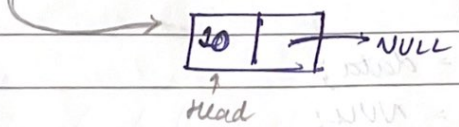
Insertion

Starting mai empty linked list hogi, aur head will be pointing to null.

```
Node * Head = NULL;
```

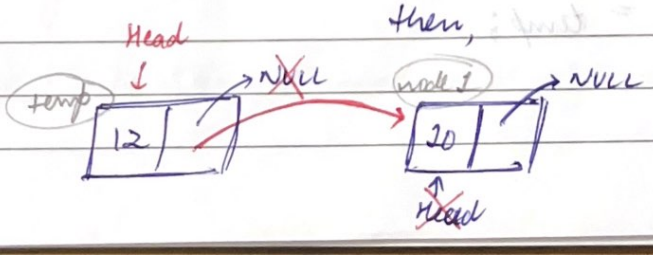
fir hum ne nai node banai :-

```
Node * node1 = new Node(10);
```



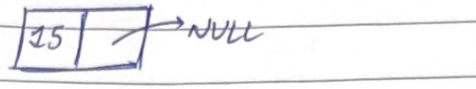
Par ek aur node daalni hai toh ek function banayenge "Insert At Head". Fir humko ek nai node banani hai with data "12".

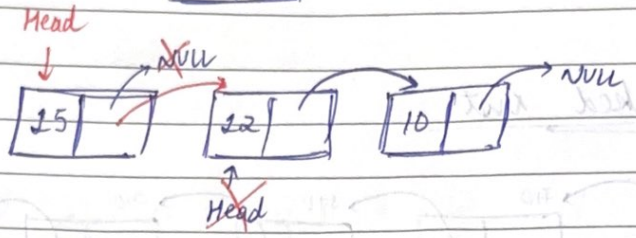
fir mujhe iss node ko node1 se pehle lagana hai



So, temp → next = ~~head~~ head / node1;
head = temp; → as head always points to starting of linked list.

Now, we want to create a new node of data = '15'

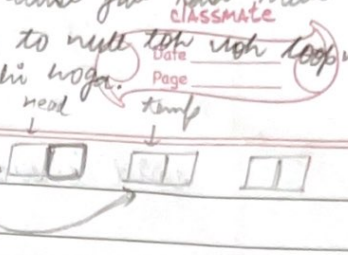




code:-

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

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

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

void InsertAtHead ( Node * &head , int d) {
    Node * temp = new node (d); // creating new node.
    temp → next = head;
    head = temp;
}
```

jaha par temp → next != NULL nhi aa sakta because jab last node par jayenge toh temp ka next is pointing to null toh woh loop me jayega hi nhi, toh last element print nhi hoga.




```
void print (node * &head){
    Node * temp = head;

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

int main (){
    Node * node1 = new Node (10); // created a new node

    Node * head = node1; // head pointed to node 1.
    print (head); → 10
    InsertAtHead (head, 12);
    print (head); → 12, 10
    InsertAtHead (head, 15);
    print (head); → 15, 12, 10

    return 0;
}
```

Ab hume chahiye ki jo bhi nayi node bane woh piche insert ho toh ek function insertAtTail banao.

insertAtTail :- ending node ke aage New node add karega.

tail :- pointer of node type, jo humesha last node ko point karega.

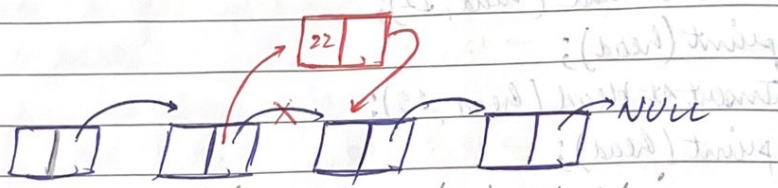
Insert at end / tail :-

```
void insertAtTail(Node * &tail, int d){

    Node * temp = new Node(d); // new node created.
    tail -> next = temp;
    tail = tail -> next;
}
```

Insert in middle :-

Jaise aisi linked list hai :-

aur 3rd position pe 22 daalna hai, then



jaun 'i' position tak jana hai toh (i-1) node tak jao.

code

```
void insertAtPosition(Node * &head, Node * &tail, int position, int d){
    Node * temp = head;
    int count = 1;

    while(count < position-1){
        temp = temp -> next;
        count++;
    }
// creating a node for d :-
Node * nodeToInsert = new Node(d);
```

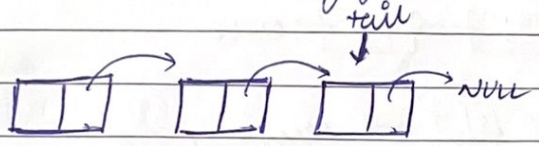
nodeToInsert->next = temp->next;
temp->next = nodeToInsert;
}

agar first position pe daalna hai toh :-

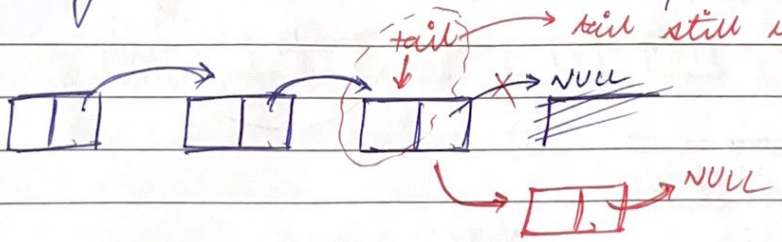
if(position == 1){
insertAtHead(head, d);
} return;

par agar last position par insert karna hai toh waha ek problem aayegi, waha tail update nhi hogi.

eg.



aur jo humne bola insert at pos" 4 then



tail updatation :- or inserting at last position

if(temp->next == NULL){
insertAtTail(tail, d);
return;
}

Delete an element :-

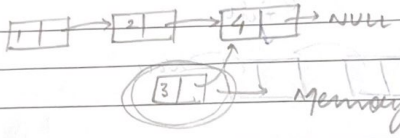
Ek linked list hai toh humne bola iss pos ko delete karado ya value daalo aur uss value to delete karado.

position

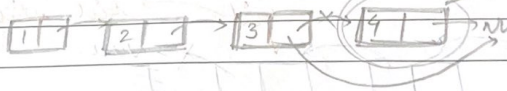
```
void deleteNode(int position, Node *&head){
```

Logic

① middle Node :-



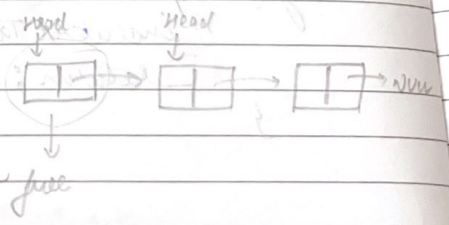
② Last node :-



③ First Node :-

↳ alag se handle karna padega because first wale case mai "previous" nhi hota.

```
if (position == 1){ // deleting first or start node
    Node * temp = head;
    head = head->next;
    delete temp;
}else{
```



temp->next = NULL;

// for deleting we need to write destructor in class.

code for constructor

```
~Node(){ // destru
    int value = this
    // memory free
    if (this->next
        delete next;
        this->next
    }
    cout << " memo
}
};
```

continue

```
}else{
    Node * curr
    Node * prev

    int count =
    while (count
        prev = cu
        current =
        count++;
    }
    prev->nex
    delete cu
}
```

code for constructor

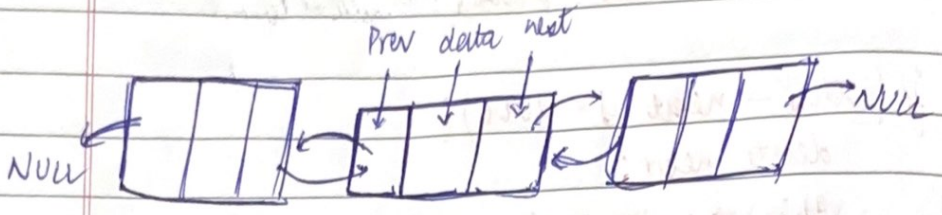
```
~Node(){        //destructor
    int value = this -> data;
    //memory free
    if( this -> next != NULL){
        delete next;
        this -> next = NULL
    }
    cout << " memory is free for Node with data " << value << endl;
};
```

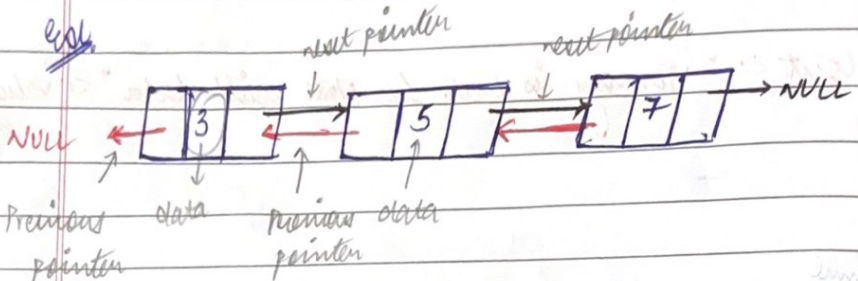
continue

```
}else{
    Node * curr = head;
    Node * prev = NULL;

    int count = 1;
    while (count < position) {
        prev = curr;
        current = current -> next ;
        count++;
    }
    prev -> next = current -> next;
    delete curr;  curr -> next = NULL;
}
```

Doubly Linked List

There is a change in structure of node :-





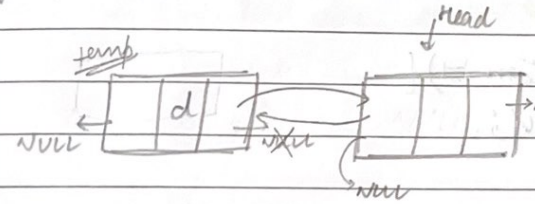
Code :-

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

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

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

→ starting mai dono null honge.

```
    cout << endl;
}

int getLength(Node * head){ // function to get length of linked list
    int len = 0;
    Node * temp = head;
    while(temp != NULL){
        len++;
        temp = temp -> next;
    }
    return len;
}
```

Logic for insert at head :-

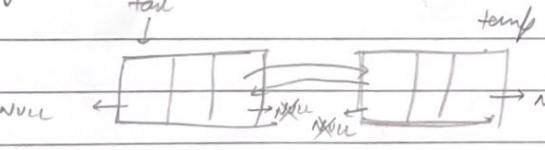


① create a new node with data "d".
② temp → next = head / head → prev
③ head → prev = temp / temp → next
④ head = temp;

Code :-

```
void insertAtHead(Node *&head, int d){
    Node * temp = new Node(d);
    temp -> next = head;
    head -> prev = temp;
    head = temp;
}
```

Logic for insert at Tail



① create temp.
② tail → next = temp;
③ temp → prev = tail;
④ tail = temp;

Code

```
void insertAtTail(Node *& tail, int d){
    Node * temp = new Node(d);
    tail -> next = temp;
    temp -> prev = tail;
    tail = temp;
}

void insertAtPosition(Node *& head, int position, int ele){
    if(position == 1){
        insertAtHead(head, d);
        return;
    }

    Node * temp = head;
    int count = 1;
    while(count < position - 1){
        temp = temp -> next;
        count ++;
    }
    if(temp -> next == NULL){
        insertAtTail(tail, d);
        return;
    }
    Node * insert = new Node(ele);
    insert -> next = temp -> next;
    temp -> next -> prev = insert;
    temp -> next = insert;
    insert -> prev = temp;
}
```

Yaaha humne koi bhi / first node nhi banaya main function then

```
Node * head = NULL;
Node * tail = NULL;
```

then empty list ko deal alag se karna padega :-

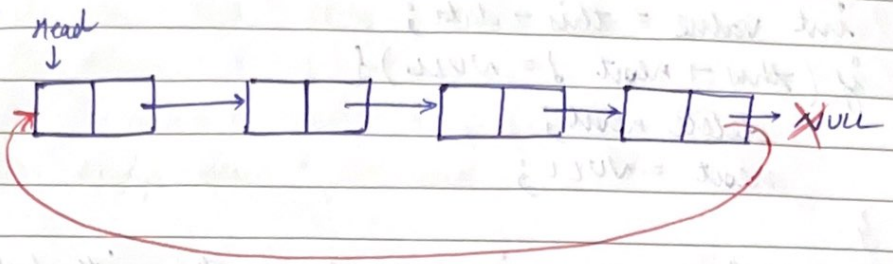
```
void insertAtHead (Node *& head, int d){
                   Node *& tail
    if (head == NULL){
        Node * temp = new Node(d);
        head = temp; tail = temp;
    } else {
        NORMAL LOGIC
    }
```

→ Same rule goes for insertAtTail.

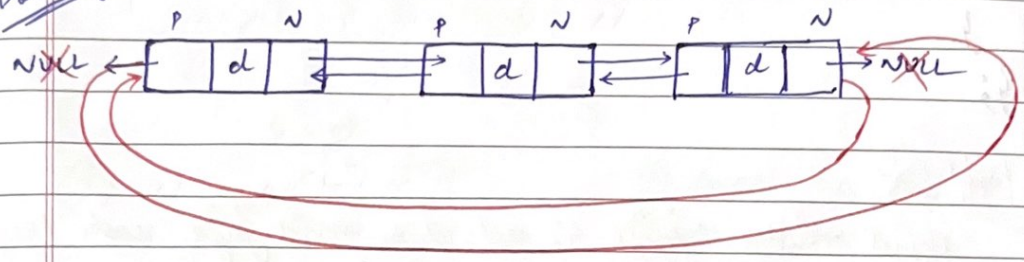
circular linked list mai head banane ki zaroorat nhi, because "tail" se hi hum sab kuch kar sakte hai.

Circular Linked List

Singly



Doubly



~~Circular Linkly Linked~~ Singly Circular Linked List :-

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

Basic structure of class remains same.

Code:-

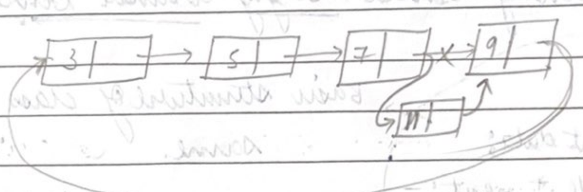
```
# include <iostream>
using namespace std;
class Node {
    public :
        int data;
        Node * next;

    Node (int d){ // constructor
        this → data = d;
        this → next = NULL;
    }
```

```
~Node() {   // destructor
    int value = this -> data;
    if (this -> next != NULL) {
        delete next;
        next = NULL;
    }
    cout << "memory is free for node with data " <<
                        value << endl;
}
};
```

Logic/approach

input = data ⇒ jaise hi woh data mil jaye uske baad ek new node daal dena of data "d".

ex.

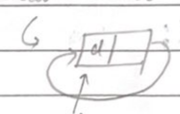


input = 7
d = 11

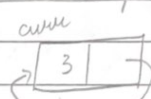
```
void insertNode (Node* & tail, int element, int d) {
```

case ① empty list (tail = NULL)

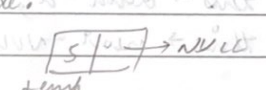
create new node of "d", tail = temp,



② 1 node is present :-



and we want to insert a node with data element "3", just create new node.



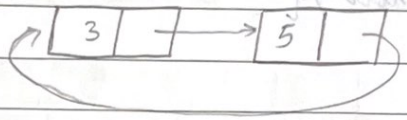
So "curr" ka next kahi-na-kahi point kar raha hoga, toh usko hum kahi store/save kar lete hai.

ⓐ forward = curr → next;

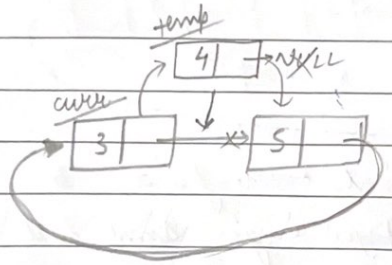
ⓑ current → next = temp;

ⓒ temp → next = forward;

③ We have 2 nodes:-



(I)

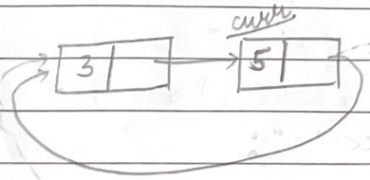


ⓐ create node

ⓑ temp → next = curr → next;

ⓒ current → next = temp;

(II)



ⓐ create node

ⓑ temp → next = curr → next;

ⓒ current → next = temp;

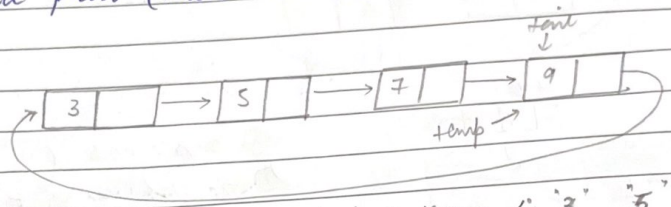
Code

```
// assuming that the element is present in list
if (tail == NULL){ // empty list
    Node * newNode = new Node (d);
    tail = newNode;
    newNodd → next = newNode;
} else { // non-empty list
    Node * curr = tail;
    while ( curr → data != element){ // for finding the
        curr = curr → next;          element in 22.
    }
```

// Now current is representing "elements" wala node
Node * temp = new Node (d); //creating new node
temp → next = curr → next;
curr → next = temp;

}
}
void print (Node * tail){

Logic



pehle "9" wale node ko print karo fir "3", "5" then "7" then rukjana, condition kya hogi same tail wale address pe pohoch jao toh rukjana. Ek address ko ek baar hi print karana.

```
Node *temp = tail;
do{
    cout << tail → data << " ";
    tail = tail → next;
}while (tail != temp);
}

int main(){
  Node * tail = NULL;

insertNode ( tail, 5, 3);
print (tail);
insertNode ( tail, 3, 5);
print (tail);
```

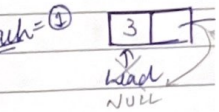
→ 3 5

insertNode (tail,
print (tail);
insertNode (tai
print(tail);

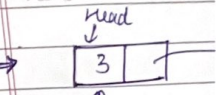
{

① Reverse a Lis

Approach = ①

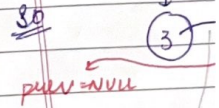


"3" → "5" ko
fir "7" ko p
head "3" ko



Node * cur

curr hum
piche ki t



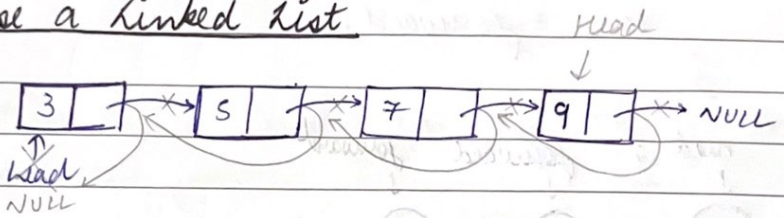
again iss
list ka
se pehle

```
insertNode(tail, 5, 7);
print(tail);  → 3 5 7
insertNode(tail, 5, 9);
print(tail);  → 3 5 9 7
```

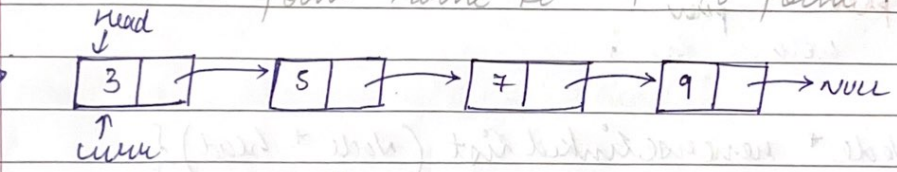
"Questions"

① Reverse a Linked List

Approach = ①



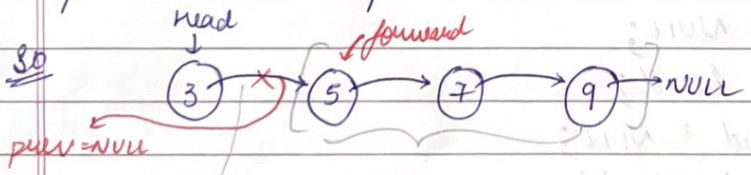
"3" → "5" ko point karne ki jagah NULL ko point kare, "5" jiska "7" ko point karne ki jagah "3" ko point kare. Fir head "3" ko point karne ki "9" ko point karado.




```
Node * curr = head;   Node * prev = NULL;
```

aur hum chahte hai ki jo bhi current node hai usko pichle ki taraah point kara do.

So



agar iss connection ko hata deta hu toh aage ki puri linked list ka path hatt jayega. Toh connection ko hata ne se pehle "5" par ek pointer lagana padega, that is "forward"

```
forward = curr → next
```