3. Linked List

- Array data structure is simple to use and array element can be accessed by A[i] where a is the name of array & i is the index or size
- Array data structure suffers some limitations

1] Size of an array is defined at the time of programming.

2] Require contiguious memory

- if the actual amount of data stored is less than the maximum size good of Amount of memory will be wasted.

⇒ Advantages of linked list

1] Linked list is the example of Dynamic Data Structure

1 They can grow & shrink during execution of progoam

efficient memory utilization

Insertion & deletion are efficient & easier.

- Introduction

The LinkedList consist of series of structures or nodes

Each structure or node consist of two field 1st data field 2nd address field. address field contant the address of its successer

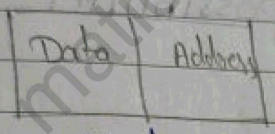


Fig. Node

A LinkedList is a collection of nodes

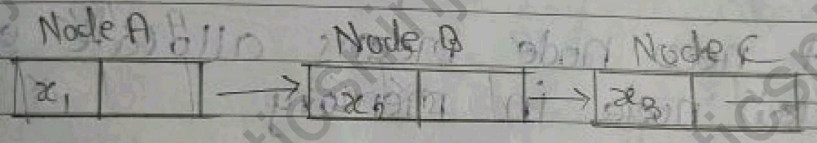


Fig: LinkedList

- Above linked list contain 3 nodes node A, node B, node C

- node A contain data x, and address of the successer node B. node B contain data x_2 & address of the successer node C, node C contain data x_3 this is last node it has no successer. there its field pointed to the null.

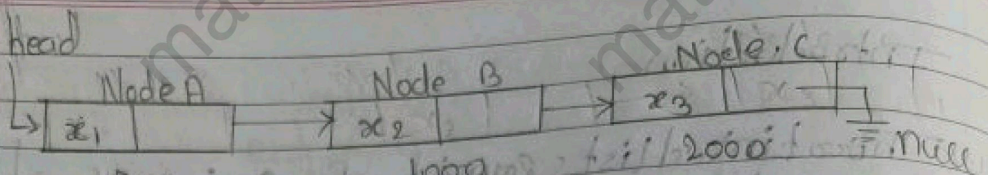
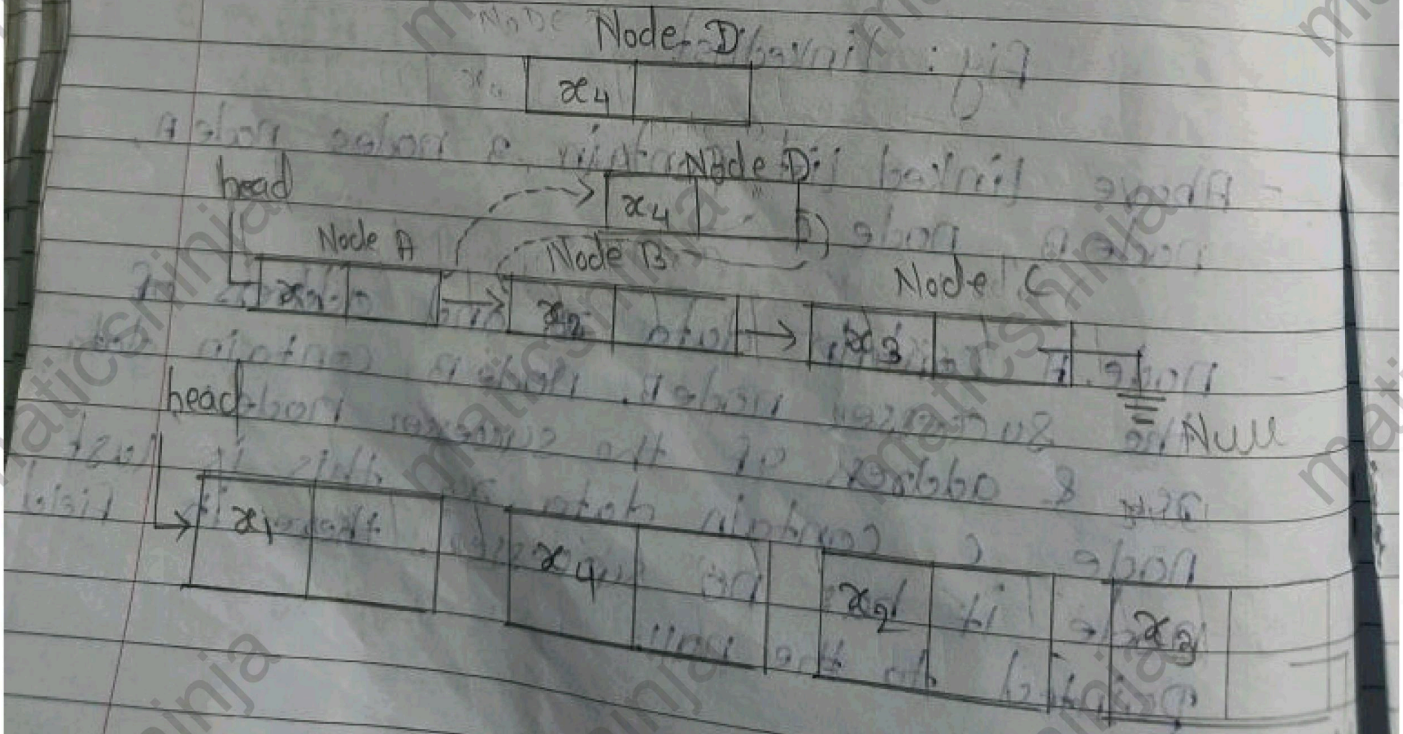


Fig : Linked list.

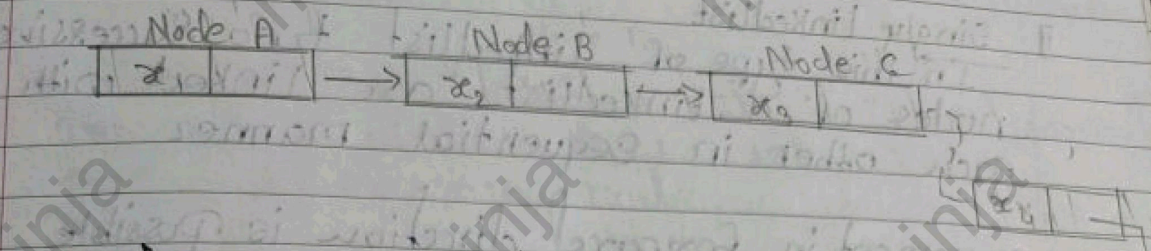
1] Insertion operation :
There are three possibilites of insertion operation

(I) At the beiginning
(II) At the middle
(III) At last

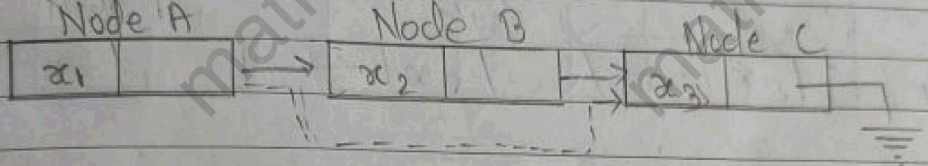
Insert new node means allocate space for new node in memory



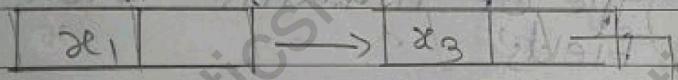
Insert new node at last / end of linked list.



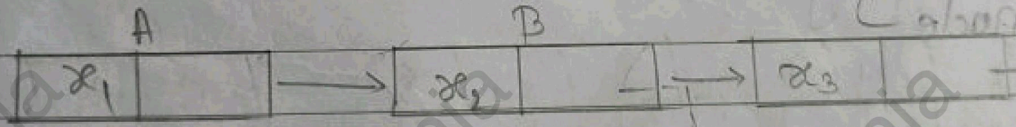
2] Deletion of a node :



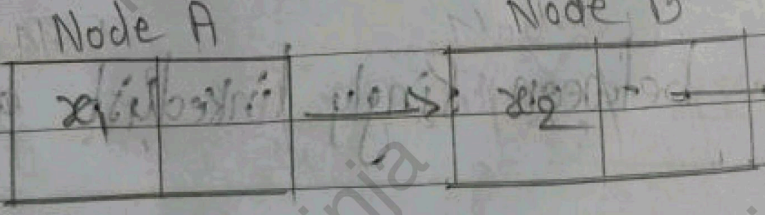
If we delete node B from above linked list



Deletion of last node :



If we delete node C then,

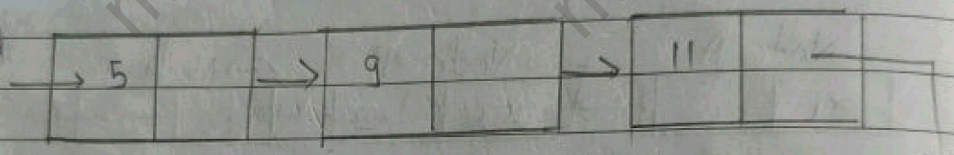


Types of Linkedlist

1] Singly linkedlist.

In this type of linkedlist two successive nodes of a linkedlist are linked with each other in sequential manner

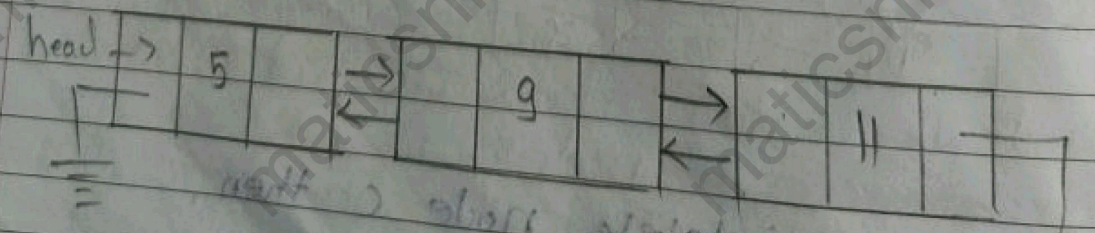
movement in forward direction is possible



2] Doubly linkedlist

In this type of linkedlist each node holds two pointer fields

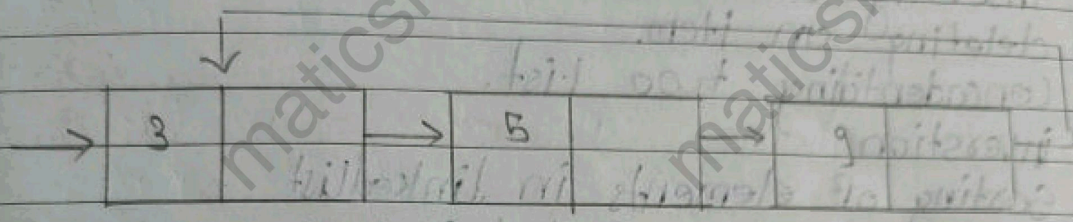
In doubly linkedlist address of next as well as preceding elements are stored with current node



Q. Distinguish between singly linkedlist and doubly linkedlist.

3] Circular Linkedlist
In circular list first and last element are adjacent.

A linkedlist can be made circular by storing the address of the first node in the next field of last node.



Q. Distinguish between linear Linkead List & circular Linkead list.

⇒ Advantages of circular linked list.

1] In circular linkedlist last node contain the address of first node. Any node can be access again & again in the loop.

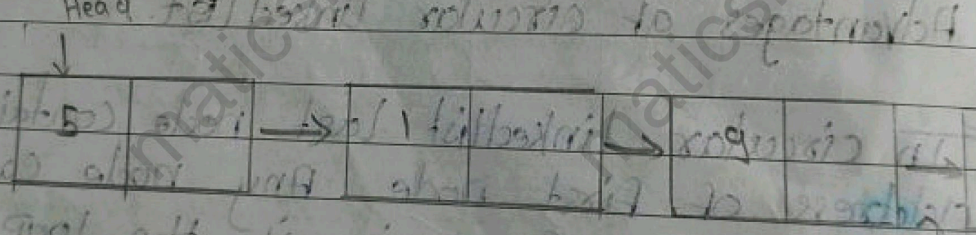
2] Traversing is possible from node to any other node in the circular linked list which is not possible in singly linked list.

3]
⇒ Application
round robbin algorithm in cpu scheduling

- Basic operations on linked list:

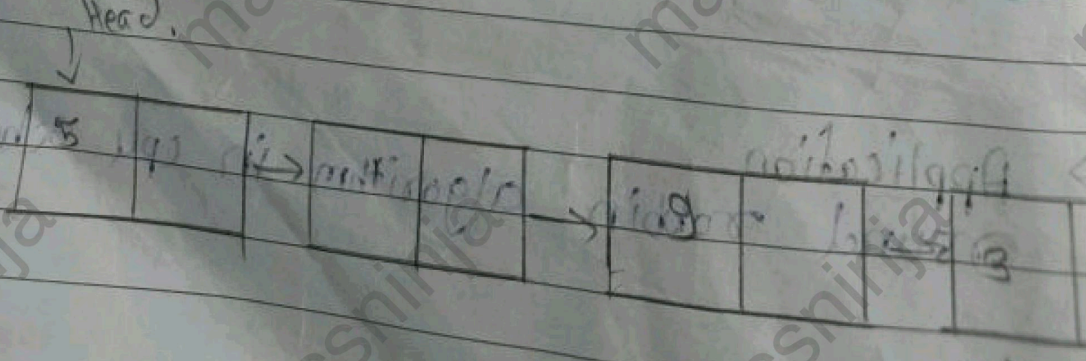
1] Creation
2] Traversing
3] printing
3] Counting the nodes in list.
4] Searching an item in list.
5] inserting an item.
6] deleting an item.
7] Concadentaing two list.
8] inversion
9] Sorting of elements in linkedlist.
10] Merging of two Sorted linkedlist.

1] Traversing :



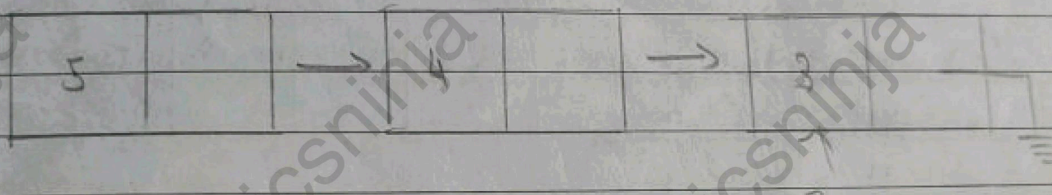
P = Head
while (P != Null)
P = P -> next

2] Counting :



```
int Count (node *p)
{
 int i;
 i=0;
while (p!=Null)
 {
 i = i+1;
 p=p→next;
 }
 return i;
 }
```

3] Printing a list through print function




```
Void Print (node *p)
{
while (p!=Null)
{
printf ("\n%d", p→data);
 p= p→next;
}
}
```

4] Insertion :

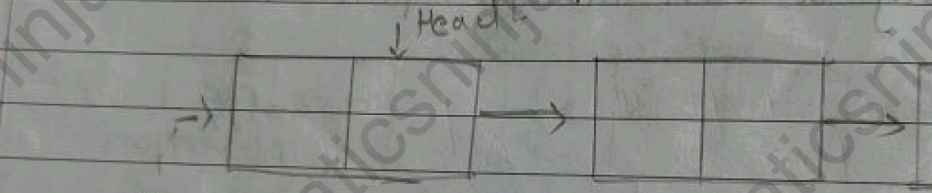
① Inserting an item at the beginning:

Step 1: obtain space for a new node.

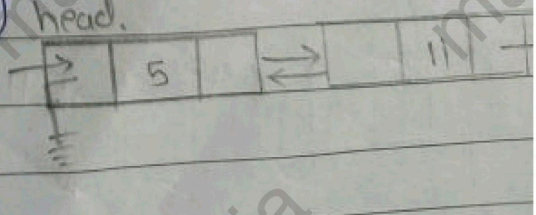
Step 2: Assign data to the data field of new node.

Step 3: Set the next field of new node to the begining of linked list.

Step 4: change reference pointer of Linked list to point new node.



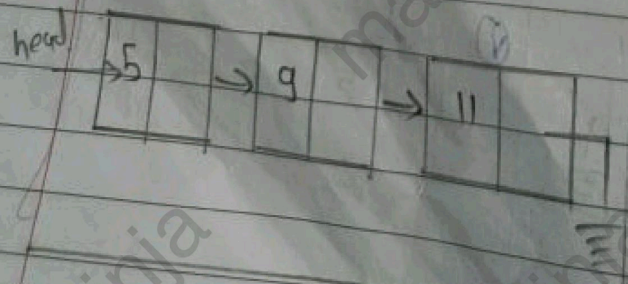
1] Distinguish between singly linkedlist and doubly linkedlist.

	Singly Linked list	Doubly Linkedlist
(i)	A linked list that contains nodes which have a data field & a next field which points to the next node in the line of nodes.	(i) A linked list that contains the data field, next field that points to the next node and a previous field that points to the previous node in the sequence.
(ii)	Allows traversing in one direction through the elements	(ii) Allows traversing in both directions backward and forward
(iii)	Requires less memory as it stores only one address	(iii) Requires more memory as it stores two address.
(iv)	Complexity of insertion and deletion at a known position is O(n)	(iv) Complexity of insertion and deletion at known position is O(1)
(v)		(v) 

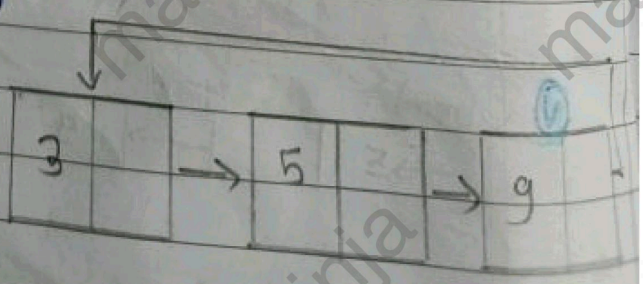
2. Distinguish between Linear Linked List and Circular linked list.

Linear LinkedList	Circular Linked list
(I) Node contain two fields in 1st part data & next part address of next node	(I) node contain two fields in 1st data & next part address of nextnode.
(II) only forward traversing is allowed	(II) only forward traversing is allowed but can jump from last node to first.
(III) It uses less memory per node (single pointer)	(III) It uses less memory per node (single pointer)
(IV) Singly linked list can mostly be used for stacks.	(IV) can be used to implement round robin method.
(V) Complexity of insertion & Deletion at known position is o(n).	(V) At known position the complexity of insertion is o(n) an deletion is o(1).

(VI)

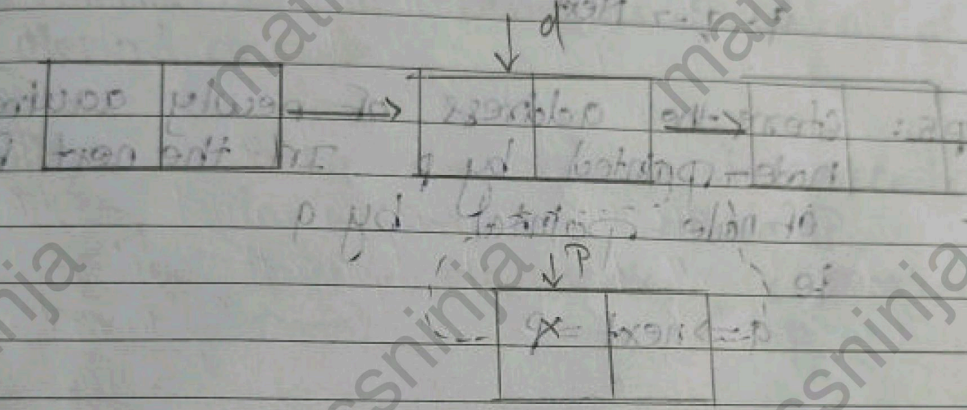


(VI)



② Insertion after node n:

Step 1 : obtain space for new node.
Step 2 : Assign value to its data field.
Step 3 : Search node N_1.
Step 4 : Set the next field of new node to point to $N_1 \rightarrow next$;
Step 5 : Set the next field of N_1 to point new node.



$p \rightarrow next = q \rightarrow next$
$q \rightarrow next = p$

③ Insertion of node at the end.

Step 1: obtain space for new node
p = (node *) malloc (size of (node))

Step 2: Assign value to its data field & make its next field
p → data = x;
p → next = null;

Step 3 : If head is Null then
head = p
goto step5

Step 4 : Position a pointer q on last node by traversing the linked list from first node until it reaches last node.
q = head
while (q → next != Null)
q = q → Next

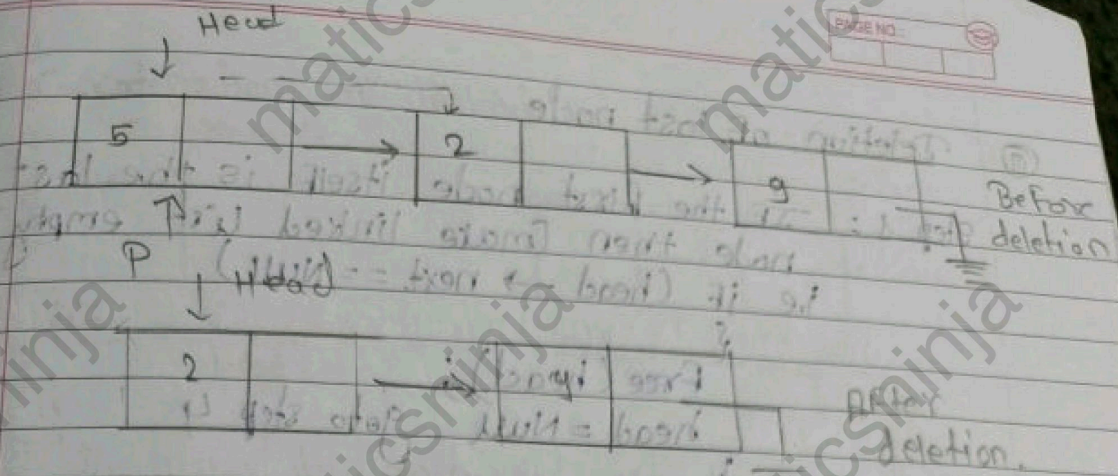
Step 5 : Store the address of newly aquired node pointed by p . In the next field of node pointed by q
ie
q → next = p

Step 6 : stop

5] Deletion

① Deleting first item

Step 1 : Store the address of first node in a Pointer Variable p.
Step 2 : move the head to next node
Step 3 : free the node whose address is stored in the Pointer Variable i.e. p.



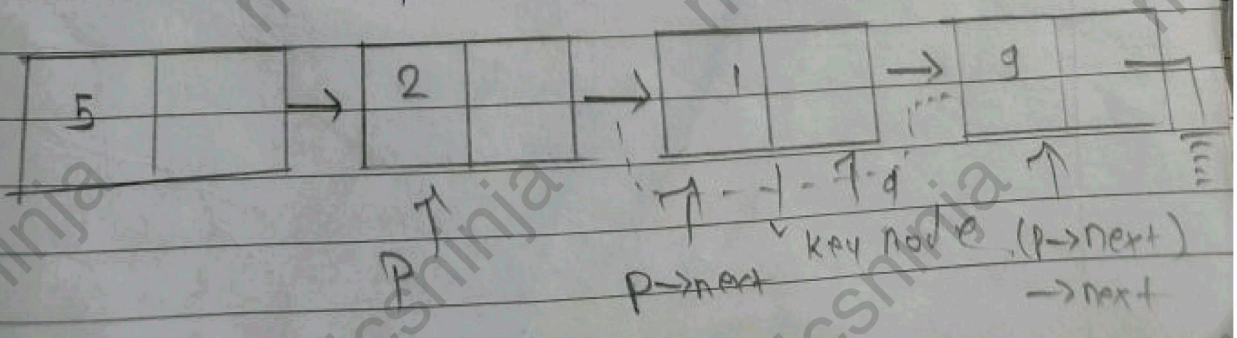
(11) Deletion of an item or Deletion of middle element or node.

Step 1 : Store the address of preceding node in a Pointer Variable P.
node to be deleted is marked as key node.

Step 2 : Store the address of key node in Pointer variable q
So that it can be free.

Step 3 : Make the Sucessor of key node as the Successor of node Pointed by P.

Step 4 : Free the node whose address is stored in the pointer Variable q.



(iii) Deletion of last node

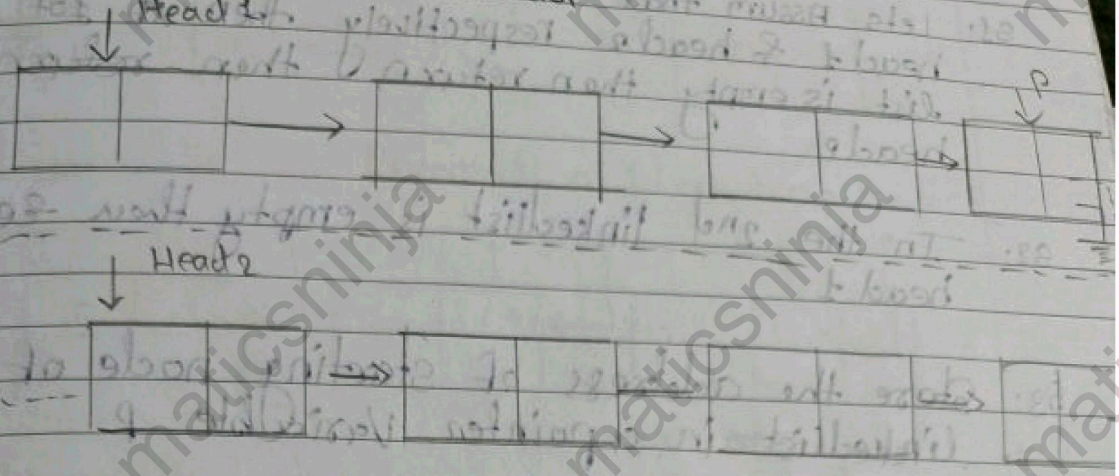
Step 1: IF the first node itself is the last node then [make linked list empty]
i.e if (head → next == Null)
{
free(head);
head = Null goto step 4
}
[otherwise]

Step 2: Position a pointer q on last but one node
q = head
while (q → next → next != Null)
q = q → next

step 3: Delete last node
P = q → next
Free(P)
q → next = Null

Step 4: Stop.

6) * Concatenation of linked list




```
Node * Concatenate ( node * head 1, node * head 2)
{
node * p;
if (head 1 == Null)
return (head 2);
if (head 1 == Null)
return (head 1);
p = head 1;
While
(p -> next != Null)
p = p -> next;
p -> next = hed2;
return (head 1);
}
```

S1: Lets Assum that two linkedlist are referrence by head1 & head2 respectively then if 1st linked list is empty then return head2

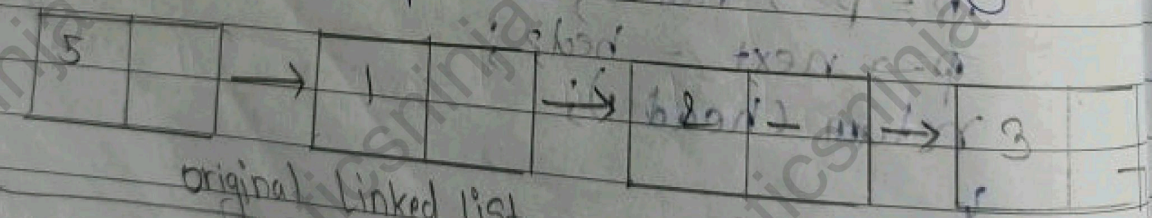
S2: In the 2nd linkedlist is empty then return head1

S3: Store the address of starting node of first linkedlist in a pointer variable p

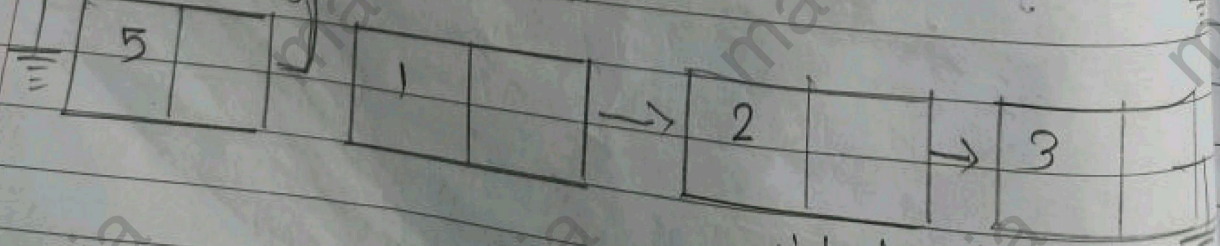
S4: move the p to the last node of linkedlist through the simple linkedlist traversal technique

S5: Store the address of 1st node of 2nd linked list in the next field of the node pointed by p.

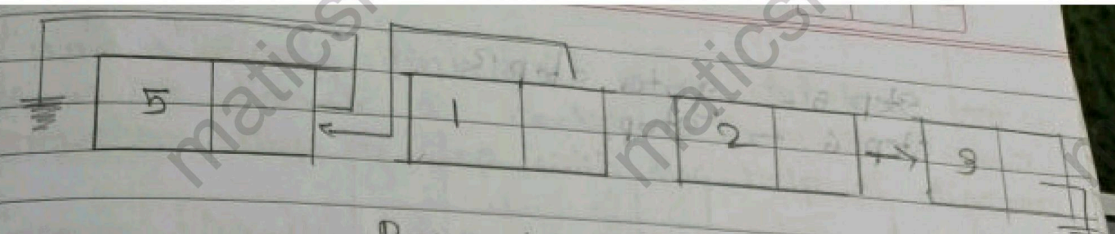
7] Inversion of linked list.
A linkedlist can be reverse by changeing the direction of the pointer



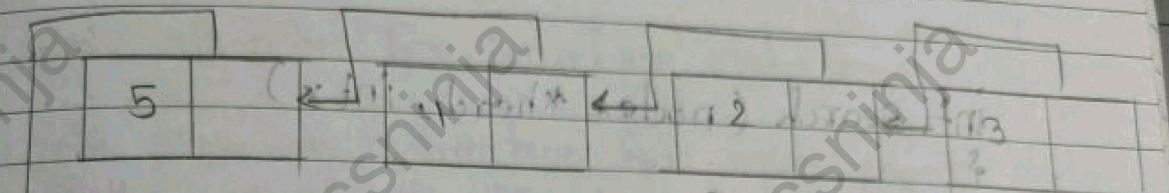
original linked list.



linked list after reversal of first node



Reversal of first two nodes



Reversal of all nodes

8) Search operation

In order to search an element in linked we start traversing a linked list from a first node

traversal end with a success if the element is found if the element is not found search end in failur.

Algorithm

Step 1 — p = head

Step 2 — If data stored in node pointed by p is x then end search with sucess ie. return(1)

Step 3 — Continue searching
p = p → next

Step 4 — If the end of linked list then return (0).

Step 5 - goto step 2
Step 6 - stop.

```
int search (node1 *head, int x)
{
 ~~int~~ node *p;
    p = head;
   while (p != Null)
 {
  if (p -> data == x)
   return (1);
    p = p -> next;
 }
  return (0);
 }
```

Write a c [...]

Applications of Linked list.

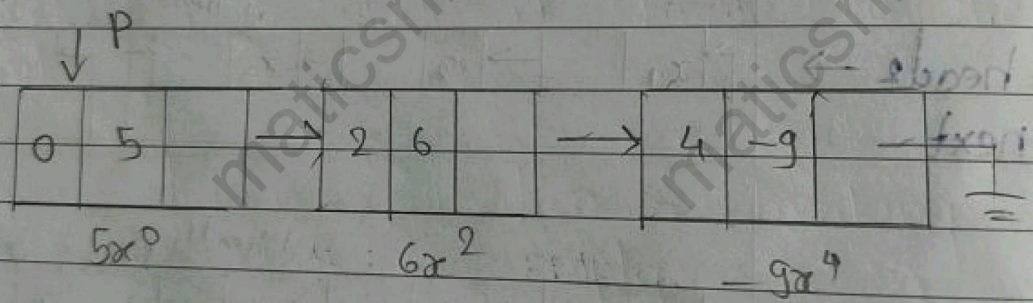
1] Polynomial as linked list

Representation of polynimal
$P(x) = a_0 + a_1 x^1 + a_2 x^2 + a_n x^4$

- n is integer value
- Each term content two tuples Power, Coefficient

$5 + 6x^2 - 9x^4$
(0,5) (2,6) (4,-9)

$5x^0 = 5x1 = 5$



2] Addition of two polynomial.

In order to add 2 polynomials denoted by P_1, P_2 whenever the powers of current terms are P_1 & P_2 are same three coefficient term added, coefficient term of new polynomial.

Ex.

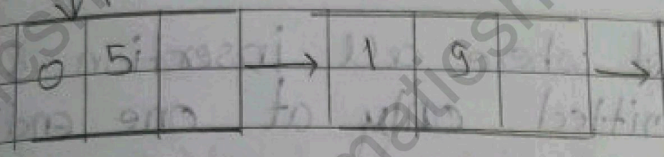
First Polynomial = $5 + 9x + 16x^2 + 13x^5 + 15x^7$ → P1

Second polynomial = $6 + 13x^2 + 9x^3$ → P2

Resultant Polynomial

$11 + 9x + 29x^2 + 9x^3 + 13x^5 + 15x^7$ → $P_1 + P_2$

P1



P2