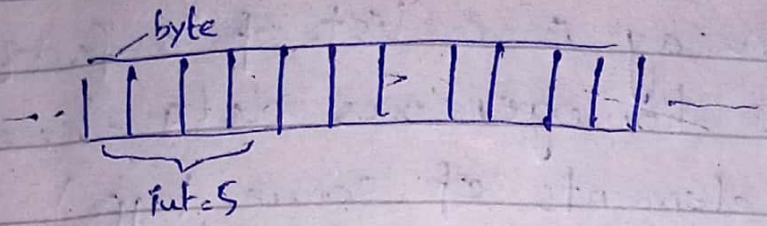
Memory: Memory is a long tape of bytes.

int a = 5; (int takes 4 bytes / 2 bytes)



Arrays: Collection of variables. (Data types should be same)

Syntax of Declaration: int a[60] ;

constant

Types of Array:

1- 1D Array
2- 2D Array
3- 3D Array

All the data will be stored in consecutive location in the memory.

Index starts from 0.

Linear Array:

"A linear array is a list of a finite number n of homogeneous data elements (i.e data elements of same type) such that:

* The elements of the array are referenced respectively by an index set consisting of the consecutive number.

* The elements of the array are stored respectively in successive memory location."

Denoted by

$A_1, A_2, A_3, \ldots A_n$

or

$A(1), A(2), \ldots A(n)$

or

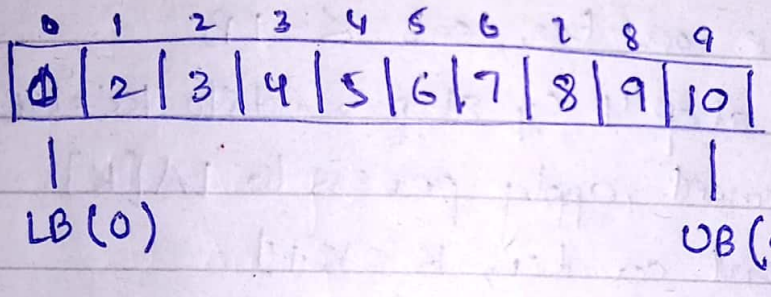
$A[1], A[2], \ldots A[n]$

linear array me finite values hti hen,
sb same data types ki hti hen,
is me hm index no. se directly access kr skte,
consecutively arranged hjaegi values sari,

Length of Array:

Length of Array = UB - LB + 1

UB = Upper Bound
LB = lower Bound



length = UB - LB + 1
= 9 - 0 + 1
= 10

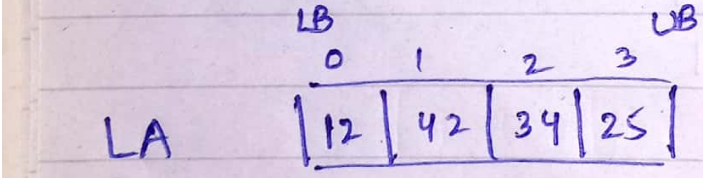
Traversing Linear Arrays:

Accessing each element of array only once so that it can be processed

⇒ Process kuch bhi hskta he jese display krna ho, 1 add krna ho, multiply krna ho ya kuch bhi, bs hm use access krnge.

Algorithm:

1) Start
2) Initialize counter, set K = LB
3) Repeat step 4 & step 5 while K<= UB
4) Visit element, apply process to LA[K]
5) Increment counter, K = K+1.
6) End.



Ye hm bubble sort or binary search me bhi use krnge.

Multidimensional Array:

	colum1	colum2	col3	col4	col5
row1	arr[0][0]	arr[0][1]	[0][2]	[0][3]	[0][4]
row2	arr[0][0]	[1][1]	[1][2]	[1][3]	[1][4]
row3	arr[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

x[10][20];

x[i][j]; (in loops).

i → row, j → col.

Introduction to Linked list

1) Single linked list (Navigation is forward only)

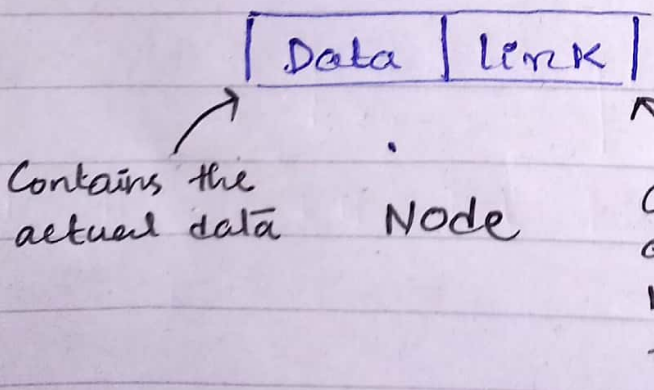
2) Doubly linked list (Forward and backward navigation is possible)

3) Circular linked list (last element is linked to the first element).

(1) Singly linked list.

A single linked list is a list made up of nodes that consists of two parts.

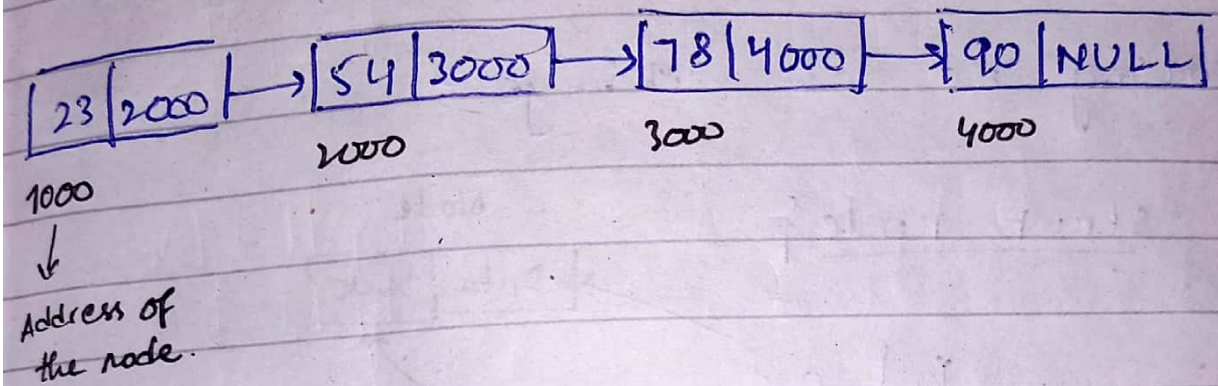
* Data
* link.



The operations we can perform on singly linked list are insertion, deletion and traversal.

Suppose we want to store a list of numbers

23, 54, 78, 90.



But how to access the first node of the linked list?

| 1000 |

Head → Pointer containing the address of that node.

Head me 1st node ka address hga, or head ek pointer he.

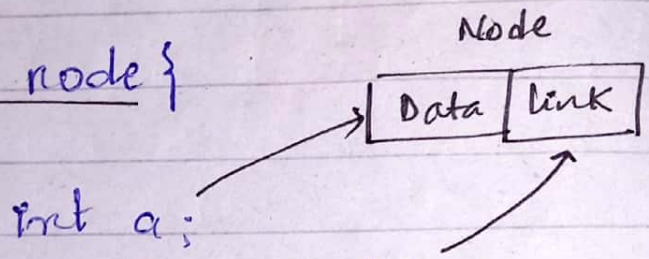
```
Struct node {
    Int data;
    Struct node* link;
}
```

⇒ Linked list me consecutive location nhi hoti he. Randomly hte hen nodes, to hm randomly acess nhi krskte kisi ko bhi.

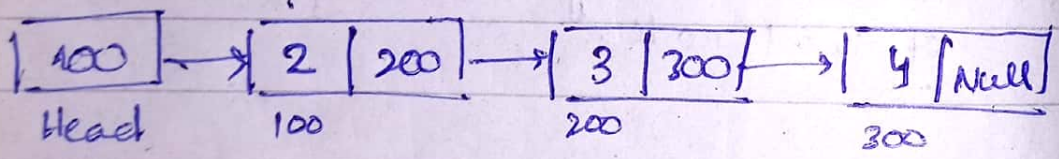
⇒ address hmesha pointers me store hta h

coding me ese likhege :

```
struct node {
    int a;
    struct node * link;
};
```



data type



If 0 nodes, then | empty list |

In this case, the last head points to null

Advantages of Linked List:

- They are dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked list reduces the access time.

Disadvantages of linked list:

- The memory is wasted as pointer requires extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in Linked list.

Application of Linked List:

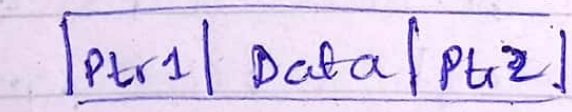
=> Linked lists are used to implement stacks queues, graphs, etc.

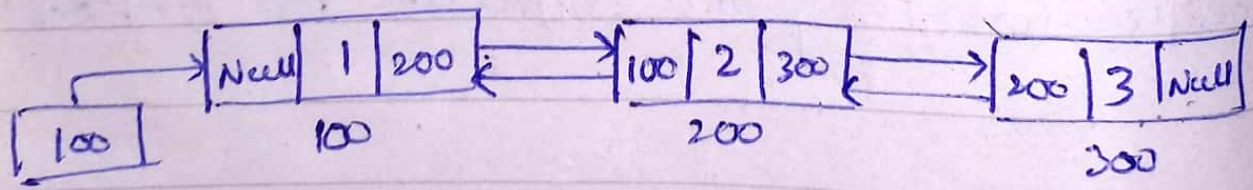
=> Linked lists let you insert elements at the beginning and end of the list.

=> In linke list, we dont need to know the size in advance.

(2) Doubly Linked List:

Each node contains a data part, and two addresses, one for the previous node and one for the next node.





Forward and backward both direction.

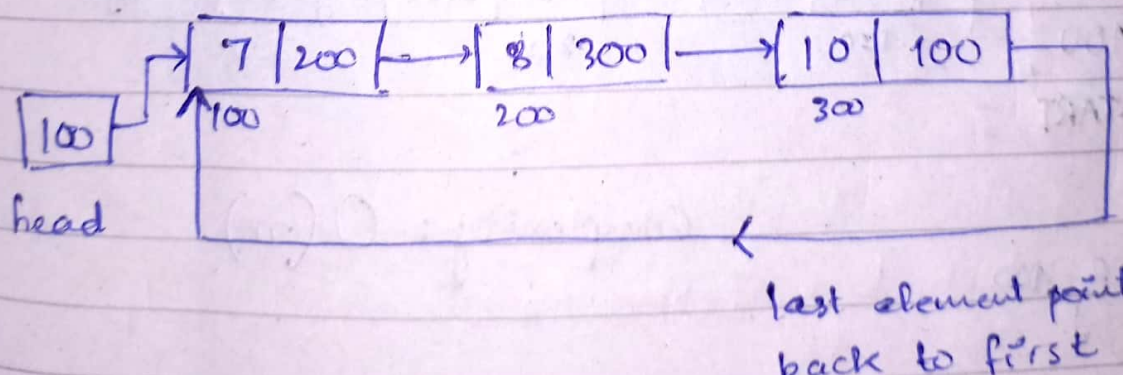
user defined data type

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

Circular linked list:

(Variation of Singly linked list)

In circular LL, the last node of the list holds the address of the first node hence forming a circular chain.



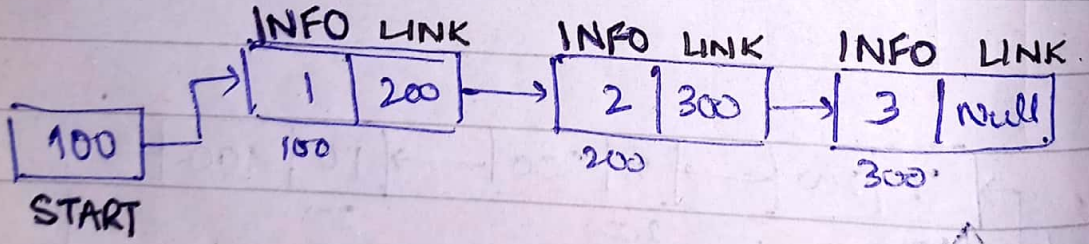
Traversing a Linked List:

Step 1: Start.
Step 2: set PTR = START (Initialise pointer PTR)

Step 3: Repeat step 4 and 5 while PTR ≠ NULL
Step 4: Apply PROCESS to INFO[PTR].
Step 5: Set PTR = LINK[PTR], (PTR now points to the next node)

End of Step 2 loop.

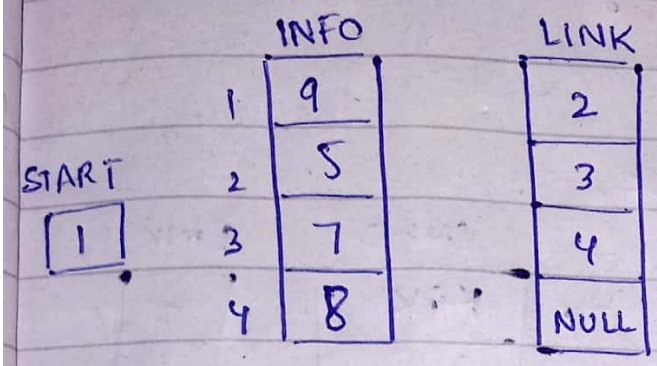
Step 6: Exit.



Complexity: O(n)

PTR = 100
↓
INFO[100] = 1
LINK[100] = 200

Searching in Unsorted linked list, complexity O(n).



Step 1: Set PTR = START

Step 2: Repeat Step 3 while PTR ≠ NULL.

Step 3: If ITEM = INFO[PTR], then:
Set PTR = NULL and Exit.
Else:
Set PTR = LINK[PTR] (Points to the next)
(End of IF Structure)
(End of Step 2 loop)

Step 4: Exit.