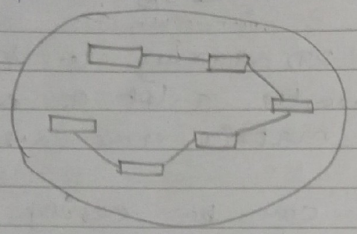
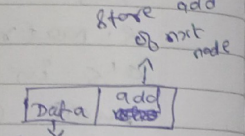
02 = LINKED LIST

* Linear D.S
* Non-contigous memory arrangemnt.
  (adjecent node-ൽ അല്ല elemnts store ചെയ്യുന്നത്)
* Dynamic memory allocation
  (ആവശ്യത്തിന് മാത്രം memory allocate ചെയ്യുന്നു.)
* A list of nodes ഉം each node has 2 parts →
  data part ഉം, add part to nxt node.





* Single node =

```
Struct node {
    int data
    Struct node *next;
}
```

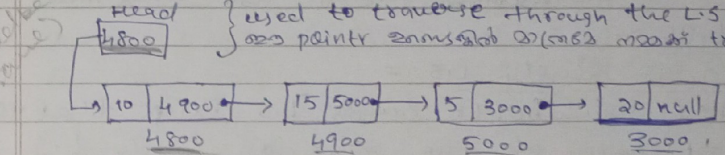
=> Array vs linked list =

Array	L.S
* Static	Dynamic
* Data in adjcnt locations	scattered in the memory pool.
* fixed size, resizing is expensive.	memory allocated as needed.
* ins & dlt need shifting	comparatively easier than array.

L.S → linked list.

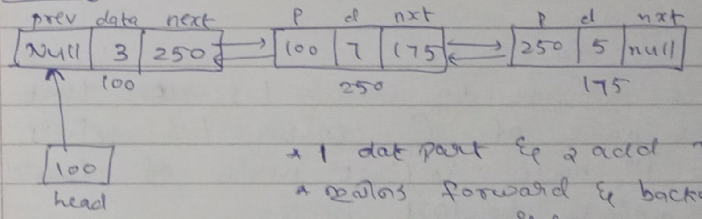
* Memory waste if array is not full	No memory waste
* Only data stored	Additional address field with each data
* Data size should know in advance	No need to know.

=> Types of L.S = (1) Singly L.S =



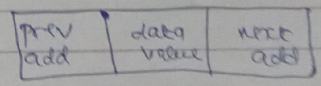
4800, 5000, 4900, 3000 } → അതിനടുത്ത് ഈ node memory-ൽ ഉള്ളത്.
+ അതിനു forward മാത്രം പോകാൻ സാധിക്കൂ.

(2) Doubly L.S =

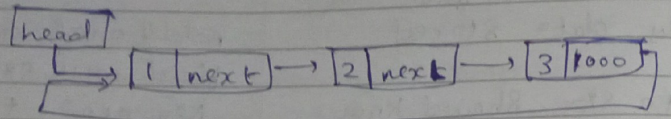


* 1 dat part ഉം 2 adrs part
* ഇതിനു forward ഉം backward മാത്രം possible.

```
Struct node {
    int data
    Struct node *next, *prev
}
```

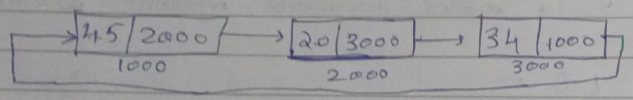


3) Circular L.S =

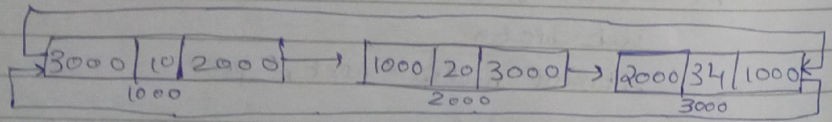


* Last elemnt will be pointed to 1st elemnt.
* use → ഓരോ list-ൽ ഉള്ള elemnts full traverse ചെയ്യാൻ traverse ചെയ്യാൻ ഇതാണ് use ചെയ്യുന്നത്.
* 2 types :-

a) Circular Singly L.S =
It is similar to Single L.S except that the last node of the circular Singly L.S points to 1st node.



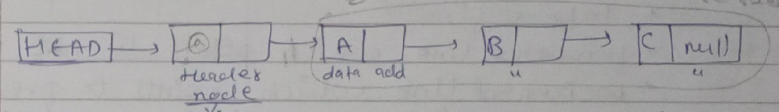
b) Circular double L.S =
It is similar to doubely L.S except that the last node of O doubely L.S points to the 1st node & first node of doubely L.S points to last node.



* Adv:
- Any node can be a starting point.
- circular doubele L.S used for implementation of advanced d.s.

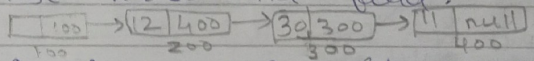
* Disadv →
- binding end of the list & loop contol is harder.

=> Header linked list =
* It is a type of L.S that has a header node at the beginning of the list.
* In a header L.S HEAD points to the header node instead of the 1st node of the list.

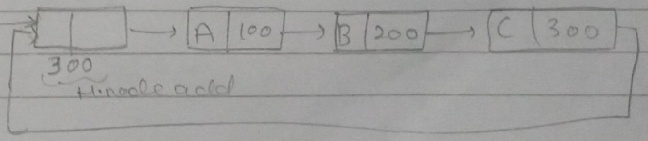


* Header node does not represent an item in the L.S.
* This data part of this node is generally used to hold any global info about the entire L.S
* next part of header node points to the 1st node in the list.
* 2 types :-

a) Grounded H.L.S = Stores NULL in the last node's nxt field.



b) Circular H.L.S = Stores the address of the header node in the nxt part of last node of the list.

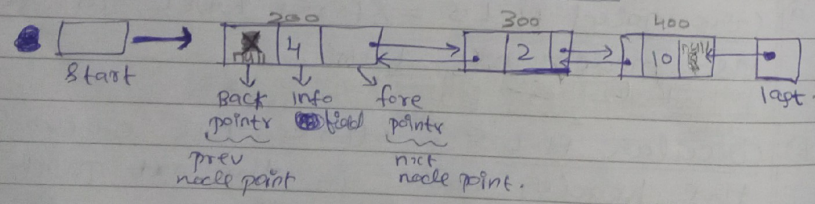


* Adv =
- null pointer is not used, & hence pointers contain valid address.
- Every node has a predecessor, so the 1st node may not require a spl case.

⇒ Two-way L.S = Doubly L.S

* It is a linear collection of data elemnts, → nodes, where each node N is divided into 3 parts –
  (a) * Info field
  (b) next node * forward link which points to nxt node.
  (c) prev node * backward link which points to prev node.
* Starting address is stored in START/FIRST pointer.
* Traverse list from end, This pointer → END/LAST.
* Every node (except last node) contains the add of nxt node, & every node (except 1st node) contains add of prev node.
* It can be traversed in either (dir).

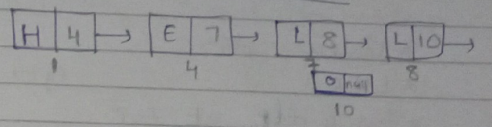
* Representation of 2-way L.S =



we can arrange HELLO in any order.

⇒ Representation of L.S in memory =

	Data	nxt
Start [1] → 1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	Null



H → 1st ∴ Start → [1].
next is E → 4 ⇒ nxt[1] = 4

Data[1] = H nxt[1] = 4
Data[4] = E nxt[4] = 7
Data[7] = L nxt[4] = 8
Data[8] = L nxt[8] = 10
Data[10] = O nxt[10] = null or -1.

⇒ Application of L.S =

* Implementation of stacks & queues.
* Representing sparse matrices.
* performing arithmetic operations on long int.
* Maintaining a directory of names.

⇒ Applications of Array =

* Implementation of stack & Queue.
* used for cpu scheduling.

* L.S creation =

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

```
int main() {
    struct node *head = malloc
        (sizeof (struct node));
    head->data = 45;
    head->link = NULL;
    return 0;
}
```