Unit - 1

Data Structure

Page No.
Date

Data structure

Data structure are a specific way of organizing data in a specialized format of a computer so that information can be organized processed, retrive quickly and effectively.

- Primitive data type

Primitive data type are the most basic data type that are used for representing simple values such as integer, float, character etc.

- Non - primitive data type

Non - primitive data type is a data type that can store the data of more than one type.
Ex- stack, queue, array, Linklist.

(#) Difference Between Primitive and Non - primitive data type

→ Primitive Data type

- Data types are pre - defined.
- PDT will have certain value
- Size depends on the type of data structure
- It can start with a lowercase
- Ex- ~~One~~ Numbers and strings.

→ Non - primitive

- It is created by programmers
- Non - primitive data types can be Null.
- Size is not fixed
- It can start with uppercase
- Ex - Array and Linked list.

(#) Linear and Non - Linear Data Structure

→ Linear Data Structure

- The data elements connect to each other sequentially. A user can transeverse each element through single run.
- Linear data structure are easier to implement.
- You can traverse linear data structure in a single run
- Memory is not utilized in an efficient way.
- In Linear data structure Single level is involved.
- Ex - Array, stack, queue.

→ Non - linear data Structure

- Data structure where data elements are not arranged sequentially or linearly are called Non - linear data structure.
- Non - linear data structure are difficult to implement.
- You can traverse non - linear data structure in multiple run
- Memory is utilized in an efficient way.
- In Non - linear data structure Multiple level is involved.
- Ex - Tree and graph.

Arrays

- An array is a linear data structure that collects element of the same data type and stores them in contiguous and adjacent memory location.
- Array works on an index system starting from 0 to (n-1) where n is the size of the array.

Ex :-

Address →	11	12	13	14
array →	A	R	R	A
index →	01	02	03	04
	↑			↑
	lower bound			Upper bound

Types of Array

* One - dimensional Array

One dimensional array contain a single row of element. There arrays are usually indexed from 0 to (n-1) where n is the size of array.

element →					
index →	0	1	2	3	4

- Syntax :- int arr[n];

* Multi - dimensional array

Multi dimensional array are a powerful data structure used to store and manage data organizationaly. This type of array consist of multiple arrays that are arranged hierarchialy. They can have any number of dimension.

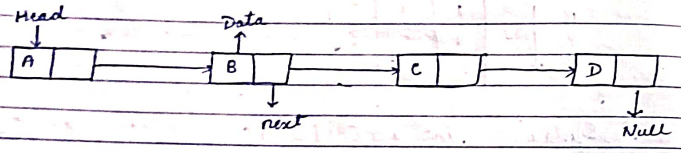
Column →		0	1	2
Row ↓	0	1	2	3
	1	4	5	6
	2	7	8	9

- Syntax :- int arr[m][n]; , int arr[m][n][o];

(#) Linked List

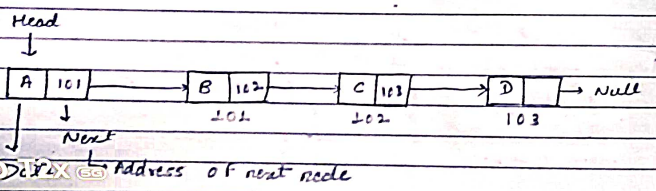
A linked list is a fundamental data structure in computer science. It consist of nodes where each node contains some data and link to next node in the sequence.

- This allow for dynamic memory allocation and efficient insertion and deletion operation.



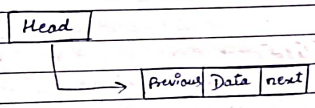
* Singly - Linked List

A singly - linked list is a linear data structure in which the elements are not stored in contiguous memory location and each node is connected to its next element using a pointer.



* Doubly - Linked List

- Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.
- Therefore a doubly - linked list a node consist of three parts
  - ↳ node data.
  - ↳ Pointer to the next node in sequence.
  - ↳ Pointer to the previous node.

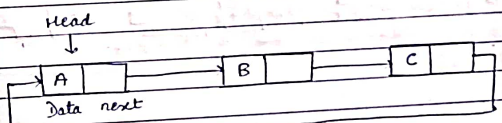


* Circular - Linked list

The Circular - linked list is a linked list where all nodes are connected to form a circle.

In a circular linked list the first node and the last node are connected to each other which form a circle.

There is no null at the End.

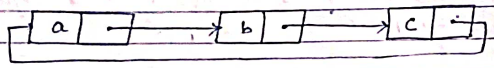


→ Singly Circular linked list

In a singly circular linked list the last node of the ~~first~~ list contains a pointer to the first node of the list.

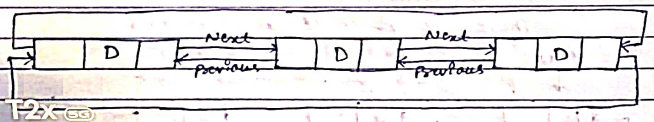
We traverse the singly circular linked list until we reach the same node where we started.

The singly circular linked list has no begining or end.



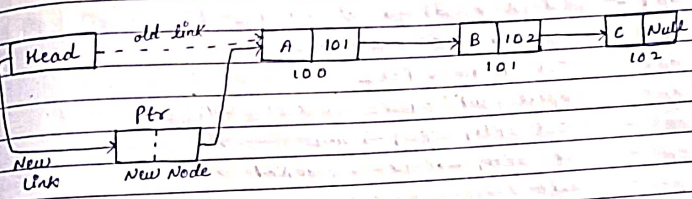
→ Doubly Circular linked list

Doubly circular linked list has all the properties of both circular linked list and doubly linked list in which two consecutive element are linked or connected by the previous or next pointer and last node points to the first node by pointer and also first node points to the last node by previous pointer.



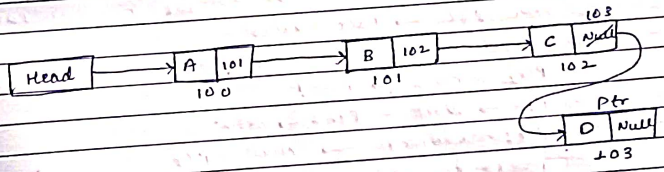
* Insertion (Single linked list)

→ At begining



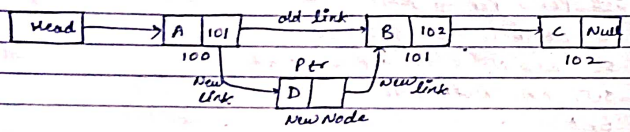
Step 1 :- If Ptr = Null (write overflow)
Step 2 :- set New Node = Ptr
Step 3 :- Set Ptr = Ptr → Next
Step 4 :- set New Node → Data = value
Step 5 :- Set New Node → Next = Head
Step 6 :- set New Node = Head
Step 7 :- Exit

→ At End



Step 1 - If Ptr = Null (write overflow)
Step 2 - Set NewNode = Ptr
Step 3 - Set Ptr = Ptr → next
Step 4 - Set NewNode → Data = Value
Step 5 - Set NewNode → Next ! = Null
Step 6 - Set Head = Ptr 1
Step 7 - Repeat until Ptr 1 → next ! = Null
Step 8 - Set Ptr, = Ptr, → next
Step 9 - Set Ptr, → next = NewNode
Step 10 - Exit

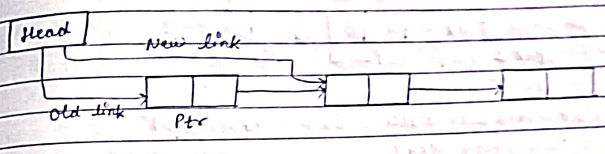
→ At specific place



Step 1 - If Ptr = Null (write overflow)
Step 2 - Set NewNode = Ptr
Step 3 - Set New Node → Data = Value
Step 4 - Set Head, = Ptr 1
Step 5 - Set Ptr 1 = Ptr 1 → next !
Step 6 - Set Ptr → Next = Ptr 1 → Next
Step 7 - Set Ptr 1 = ~~New~~ Node → Next = Ptr
Step 8 - Set Ptr = New Node
Step 9 - Exit

* Deletion

→ At begining

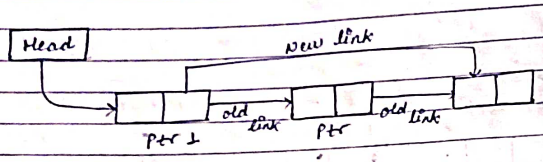


Step 1 - If Head = Null (write underflow)
Step 2 - Set Ptr = Head
Step 3 - Set Head = Head → Next
Step 4 - free Ptr
Step 5 - Exit

→ At End

Step 1 - If Head = Null (write underflow)
Step 2 - Set Ptr = Head
Step 3 - Repeat until Ptr → Next ! = Null
Step 4 - Set Ptr 1 = Ptr
Step 5 - Set Ptr 1 → Next = Null
Step 6 - free Ptr
Step 7 - Exit

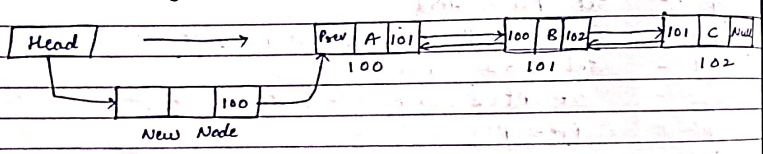
→ At Specific place



Step1 - If Head = Null (write underflow)
Step2 - Set Ptr = Head
Step3 - Set Ptr = Ptr → Next
Step4 - Set Ptr ⊥ = Ptr
Step5 - Set Ptr → next = Ptr ⊥ → Next
Step6 - free Ptr
Step7 - Exit

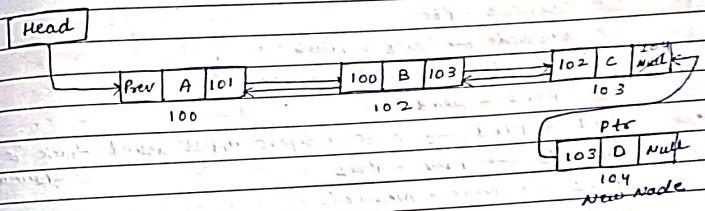
* Insertion (Doubly linked list)

→ At begining



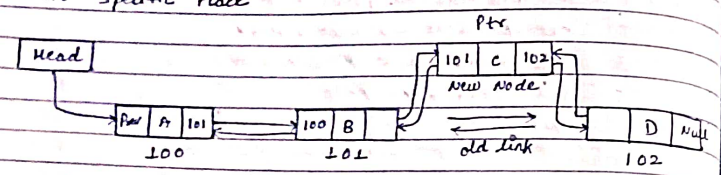
Step1 - If Ptr = Null (write overflow)
Step2 - Set New Node = Ptr
Step3 - set New Node → Data = value
Step4 - set New Node → Prev = Null
Step5 - set New Node → next = Head
Step6 - Set Head → prev = New Node
Step7 - Set Head = New Node
Step8 - Exit

→ At End



Step1 - If Ptr = Null (write overflow)
Step2 - Set New Node = Ptr
Step3 - set New Node → Data = value
Step4 - set New Node → Next = Null
Step5 - set Ptr ⊥ = Head
Step6 - set Ptr = Ptr ⊥ → Next ! = Null
Step7 - Set New Node → Prev = Ptr ⊥
Step8 - Set Ptr ⊥ → next = New Node
Step9 - Exit
Step10

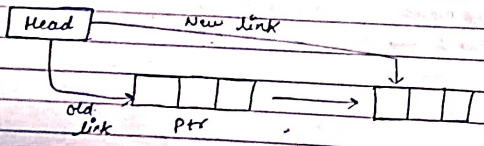
→ At specific Place



Step1 If Ptr = Null (write overflow)
Step2 set NewNode = Ptr
Step3 set NewNode → Data = value
Step4 set New Node = New Node → Next
Step5 set Ptr1 = Head
Step6 Ptr1 = Ptr1 → Next (Repeat untill desired track is found)
Step7 NewNode → Prev = Ptr1
Step8 Ptr1 → Next = NewNode
Step9 Ptr1 → next → Prev = New Node
Step10 New Node → Next = Ptr1 → next
Step11 End

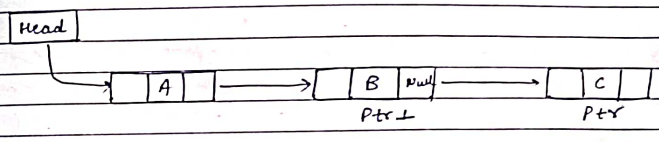
* Deletion

→ At begining



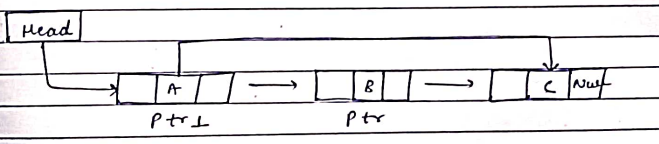
Step1 - If Head = Null (write underflow)
Step2 - set Ptr = Head
Step3 - set Head = Head → Next
Step4 - set Head → prev = Null
Step5 - free Ptr
Step6 - Exit

→ At End



Step1 - If Head = Null
Step2 - set Ptr = Head
Step3 - Ptr → Next = Null (Repeat Untill)
Step4 - set Ptr1 = Ptr1 → Null
Step5 - set Ptr1 → ~~Ptr~~ Prev = Null
Step6 - free Ptr
Step7 - Exit

→ At Specific Point



Page No.
Date

Step 1 - If Head = Null
Step 2 - Set Head = Ptr
Step 3 - set Ptr = Ptr → next
step 4 - Set Ptr1 = Ptr → next
step 5 - set Ptr → next → prev = Ptr1
Step 6 - set Ptr1 → next = Ptr → next
Step 7 - free Ptr
Step 8 - Exit