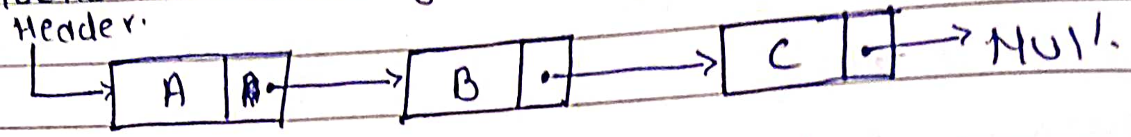
Module-3 Q.B.

1. What is linked list ? State different types of linked list.

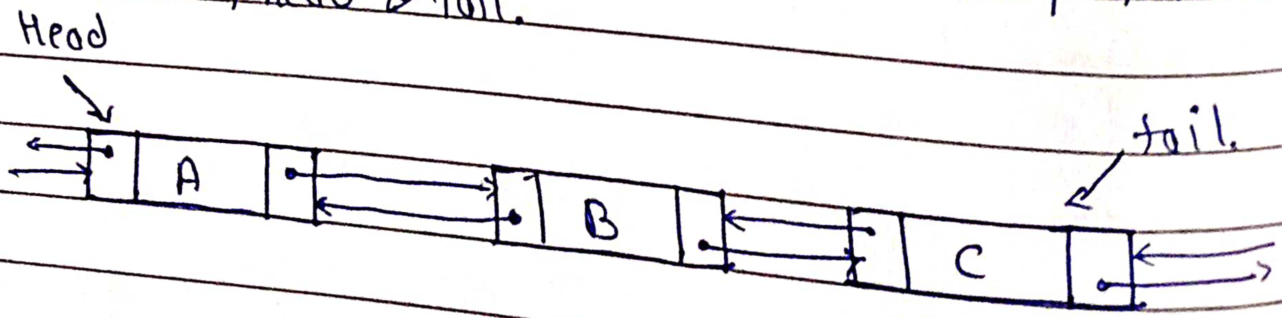
→ A linked list is a data structure which allows you to store data dynamically and manage data efficently.
Typically, a linked list, in its simplest form like looks like following:



Types of lists:-

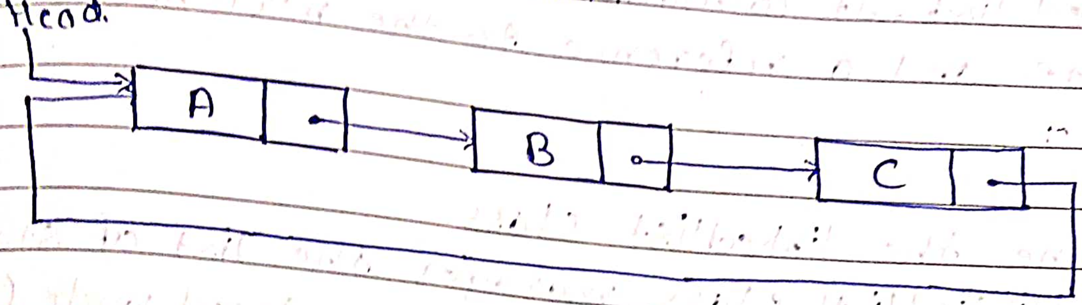
i] Single linked list (or simply linked list)
A head pointer addresses the first element of the list. Each element points at a successor element. The last element has a linked value Null.

ii] Double linked list:-
- Pointers exist between adjacent nodes in both directions.
- The list can be traversed either forward or backword.
- Usually two pointers are maintained to keep track of the list, head & tail.



Page No.
Date

iii] Circular linked list
The pointer from the last element in the list points back to first element.



2. Explain advantages of linked list over array.
→
- In linked list adjacency between any two elements are maintained by means of links or pointers.
- It is essentially a dynamic data structure.
- linked list are suitable for
  - Inserting an element at any position
  - Deleting an element from anywhere
  - Applications where sequential access is denied.
  - In situations, where the number of elements cannot be predicted.

3. Consider an information management system that manages data of students (fields rollno name). Apply suitable concepts of linked list and write an algorithm to insert a data record at the end of the list.
→ To implement an information management system that maintains student data using linked list, you can create a 'node' class to represent each student and a 'linkedlist' class to manage the list of students. The insertion algorithm will add a new student's data at the end of the linked list.

i] Define the Node class
The 'Node' class represents each student in the linked list. It cointains the student's roll number, name. and a reference to the next node in the list.

ii] Define the linkedlist class.
The 'linkedlist' class manages the list of students. It cointains a reference to the head ~~node~~ (first node) of the list and methods to insert a new node at the end.

iii] Algorithm to insert a record at the end
The algorithm to insert a data record at the end of the linked list as follow:-

- Create a New node:
  Initialize ~~to~~ a new node with the provided roll number and name.
- Check if the list is empty:
  If the list is empty (i.e. the head is 'None).
  set the head ~~to~~ to the node.
- Traverse the list:
  If the list is not empty, start from the head and traverse to the end of the list by following the 'next' pointers.
- Insert the New node:
  Once the end of the list is reached (i.e. 'current.next' is 'None'). set the 'next' pointer of the last node to the new node.

Page No.
Date

4. Write an algorithm to implement insertion, deletion, transversal on singely linked list.
→ —

5. Explain the data structure that is capable to efficently utilizes holes in memory while loading data.
→ The data structure capable of effeciently utilizing holes in memory while loading data is typically a linked list.
• linked list overview:
A linked list is a linear data structure where elements, called nodes, are stored non-contingously in memory. Each node cointains two parts:
a] Data : The actual value or data stored in the node.
b] Pointer (Next) : A referenle to next node in the sequenle.

◦ Efficent utilization of memory holes:

i] Non-Contiguous Storage :
Unlike arrays, where memory must be allocated contiguously linked list can utilize non-contiguous memory blocks (holes). This means that as long as there is enough free memory scattered across the system, a linked list can be built without needing large contiguous block of memory.

ii. Dynamic memory allocation:
Nodes in a linked list are typically allocated dynamically at runtime. If there is a small hole of free memory, a node can be allocated there, with the pointer linking it to the next node, which might be in a completely different location in memory.

iii] Flexible size:
Since nodes can be scattered, linked list can grow and shrink as needed without worrying about the size constraints that typically apply to arrays.

- Use cases:

i] Memory Management system:
In systems where memory fragmentation is an issue, linked list can be used to keep track of free blocks of memory, efficently allocating and deallocating memory as needed.

6. Sketch the process of insertion at middle in a singly linked list.

→ To sketch the process of insertion at the middle of a singly linked list. let's walk through the steps:

i] Define the singly listed structure.
- Node: Each node cointains 'data' and a 'next' pointer to the next node.

ii] Singly linked list: The 'list' has a 'head' pointer that points to the first node in the last.

ii] locate the Insertion-point:
- step 1: Traverse to the middle
  - Use two pointers, 'slow' and 'fast'.
  - Start both pointers at the 'head'.
  - Move 'slow' one step at a time and 'fast' two steps at a time.
  - When 'fast' reaches the end of the list, 'slow' will be at the middle.

iii] Insert the new node.
- step 2: Create a new node
  - Create a new node 'new_node' with the desired 'data'.
- step 3: Adjust pointers
  - Point the 'next' of 'new_node' to 'slow.next'.
  - Update 'slow.next' to point to 'new_node'.

iv] Update the list
- The list ~~has~~ now has the new node inserted in the middle, and 'next' pointers are updated accordingly.

Visualization.
Initial list: head → [1] → [2] → [3] → [4] → [5] → none.

1. Traverse

$$\begin{tikzcd}
\text{slow} \arrow[r] & {[1]} \arrow[d] & \text{fast} \arrow[r] & {[1]} \arrow[d] \\
\text{slow} \arrow[r] & {[2]} \arrow[d] & \text{fast} \arrow[r] & {[3]} \arrow[d] \\
\text{slow} \arrow[r] & {[3]} & \text{fast} \arrow[r] & {[5]}
\end{tikzcd}$$

Now slow is at the middle (node with value 3).

2. Insert :
Create a new node with value '10'.:
new_node → [10]
3. Adjust pointers:

Slow → [3] → [4] becomes Slow → [3] → [10] → [4]

Final list :
head → [1] → [2] → [3] → [10] → [4] → [5] → none.

7 —