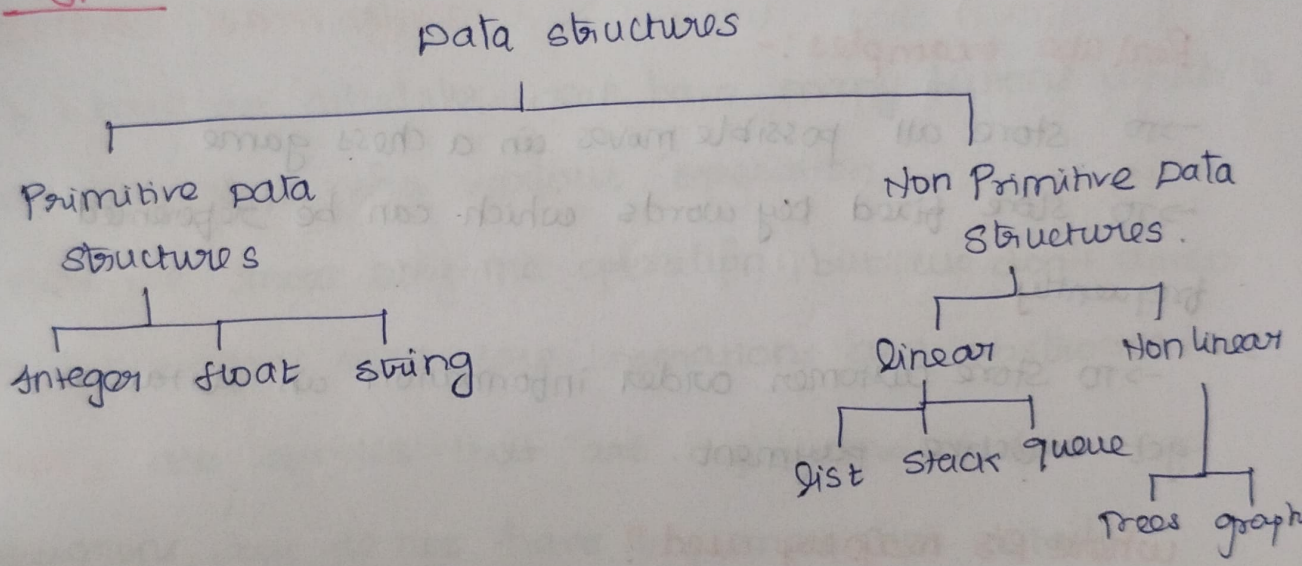
UNIT-I

LINEAR STRUCTURES AND TREES

Introduction :-

Data structure is a way of storing, organizing and retrieving data in a computer. so, that, data can be used in an effiient manner. Also, it provides a way to manage large amount of data.

Types:-



Non Primitive Data structures:

These are derived from primitive data structures. This defines the group of homogeneous and non-homogeneous data items.

Ex: Arrays, lists, graphs, trees

Primitive Data Structures:-

→ It is a basic data structure

→ It directly operated upon machine instructions.

Ex: Int, char, float.

What is the importance of data structures?

-> DS provides a right way to organize the information in the digital space to analyze all larger data.

-> This is the key component of CS in AI, OS, graphics etc

-> It is the main building block for software development process. which incorporates all s/w languages

-> In IT, it is used for data processing, automated reasoning and calculation & increases problem solving ability

Real life examples :-

-> To store all possible moves in a chess game

-> To store fixed key words which can be referenced frequently.

-> To store customer order information in Restaurant to get collective payment.

Where DS majorly used?

-> DS is the unique way of storing or organising data in Computer memory.

-> AI, OS, DS, Big data, Compiler design etc.,

Applications of DS :.

Search - To search an item in DS

Sort - Algorithm to sort an item in certain order.

LINKED LISTS

Linear Data structure:-

→ Here data are arranged in sequential order.

Ex: Arrays, linked lists, stacks, queues.

Non-Linear Data structure:

→ Here data are arranged in hierarchical order.

Ex: Trees and graphs.

ABSTRACT DATA TYPES:

If i have an calculator, i can have many buttons which is used for performing various operation.

Here we know only the operation, but we don't know how operations take place. operations are abstracted here.

ADT's are entities that are definitions of data, and operations, but do not have implementation details.

→ It is a set of operations., mathematical abstractions.

Ex: LIST ADT, set ADT, graph, tree ADT's.

Adv:

→ modular programming - all functions are independent to each other.

LIST ADTS

→ Array - collection of elements stored in consecutive memory location. when we are not sure of number of elements in advance, the memory usage is not efficient one.

->when elements are stored in random, then only memory usage is efficient.

->Hence there is need for data structures to remove above mentioned issues.

Linked list is the best solution.

->This does not store the elements in consecutive locations as user can add any number of items to it.

->But in an array, there is random access concepts. but linked list does not have random access. list can be accessed in sequential manner.

Linked list:-

-> Linear collection of data elements.

-> data elements are called as node.

->This is the building block of stacks, queues.

N - size of list

A_1 - 1st element

A_N - last element

$A_1, A_2, A_3 \ldots A_{i-1}, A_i, A_{i+1}$.

Insert:- (x, 4) means insert x after 4.

Delete:(x) - delete x from list.

find (x) - Returns position of x

next(i) - i next i+1 returns

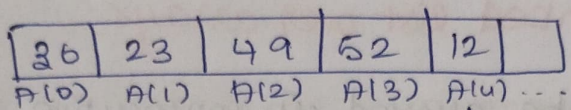
previous(i) - previous i returns

print list() - make empty()

Three types of list implementation:

1. Array based implementation
2. Linked list
3. cursor based.

Array:



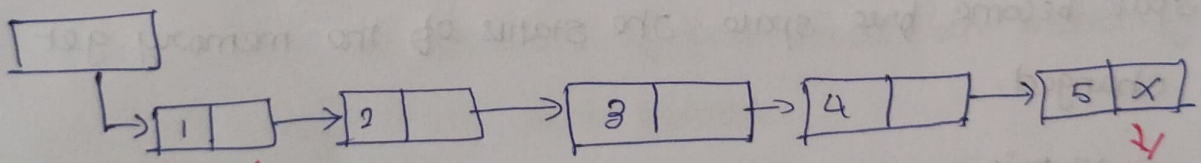
-> ordered collection of elements.

Operations:

-> Insertion.
-> deletion
-> print list : Traversal- visiting all elements in an array.
-> Find

Linked list: -> also called self referencial data type.

-> Linear collection of data elements
-> Data elements are called as node.
-> This is the building block to stacks, queues etc...



integer part

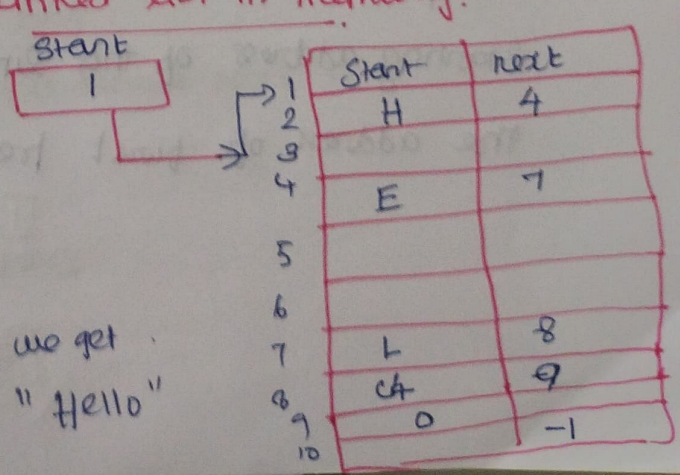
address part → pointer to next node.

Null value. (-1 in Pgm)

Code:

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

Linked list in memory:



we get "Hello"

This fig has chunk of memory location where empty hold data of other application.

Here data are consecutive memory location. 1, 4, 7, 8, 10 in this case.

Advantages of linked list over arrays:-

-> linked list can be accessed in consecutive memory location and it can accessed in sequential manner.

-> linked list cannot have random access.

-> only insertion and deletion can be done at end.

-> There is no restriction of number of nodes, as like an array.

Memory allocation and de allocation for linked lists:-

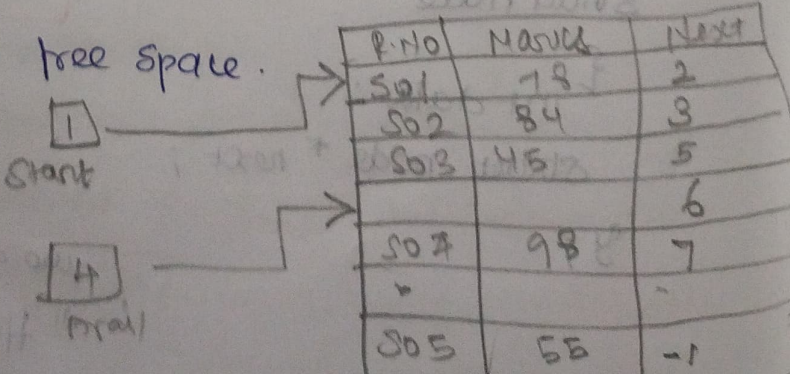
If we want to add node to an already existing linked list in the memory, we have to find a free space in the memory and then use it to store the information.

->Thus if we delete a data means, Then available & occupied space become free space. The status of the memory get changed.

-> All this memory management is done by the OS.

->System maintains a list of free cells called free pool.

-> A separate list is maintained for this free pool. The starting address of this linked list is "Avail" which denotes the address of first free space.



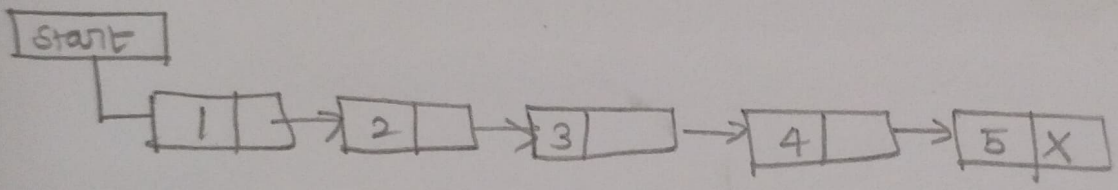
->when a node is getting deleted node means, then that space must be provided to free space.

-> This operation is done by cpu, whenever it finds free empty time or cpu is devil. or the time or task requires memory space.

->The OS must scan through memory cells and mark those cell that are being used by some other programs. Then add that free cells to free space pool.

SINGLY LINKED LISTS:-

->This is simple linked list which has data and pointer to the next node.



Traversing:-

Traversing means accessing the nodes to do some processing.

START - stores the address of first node in memory.

END Node - marked as Null pointer.

Algorithm:-

S1: Initialize SET PTR = START -> denotes 1st node a list.

S2: Repeat step S3 and 4 while PTR != NULL -> while loop until last node.

S3: PTR -> DATA -> to print 1st data

S4: Set PTR = PTR -> NEXT -> this denote next data.

End of loop.

S5: EXIT.