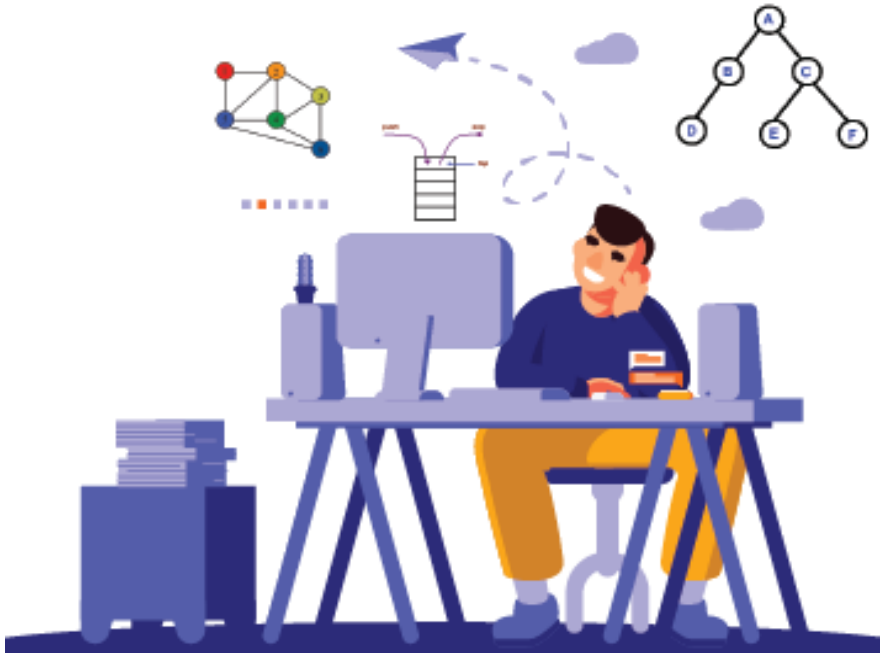
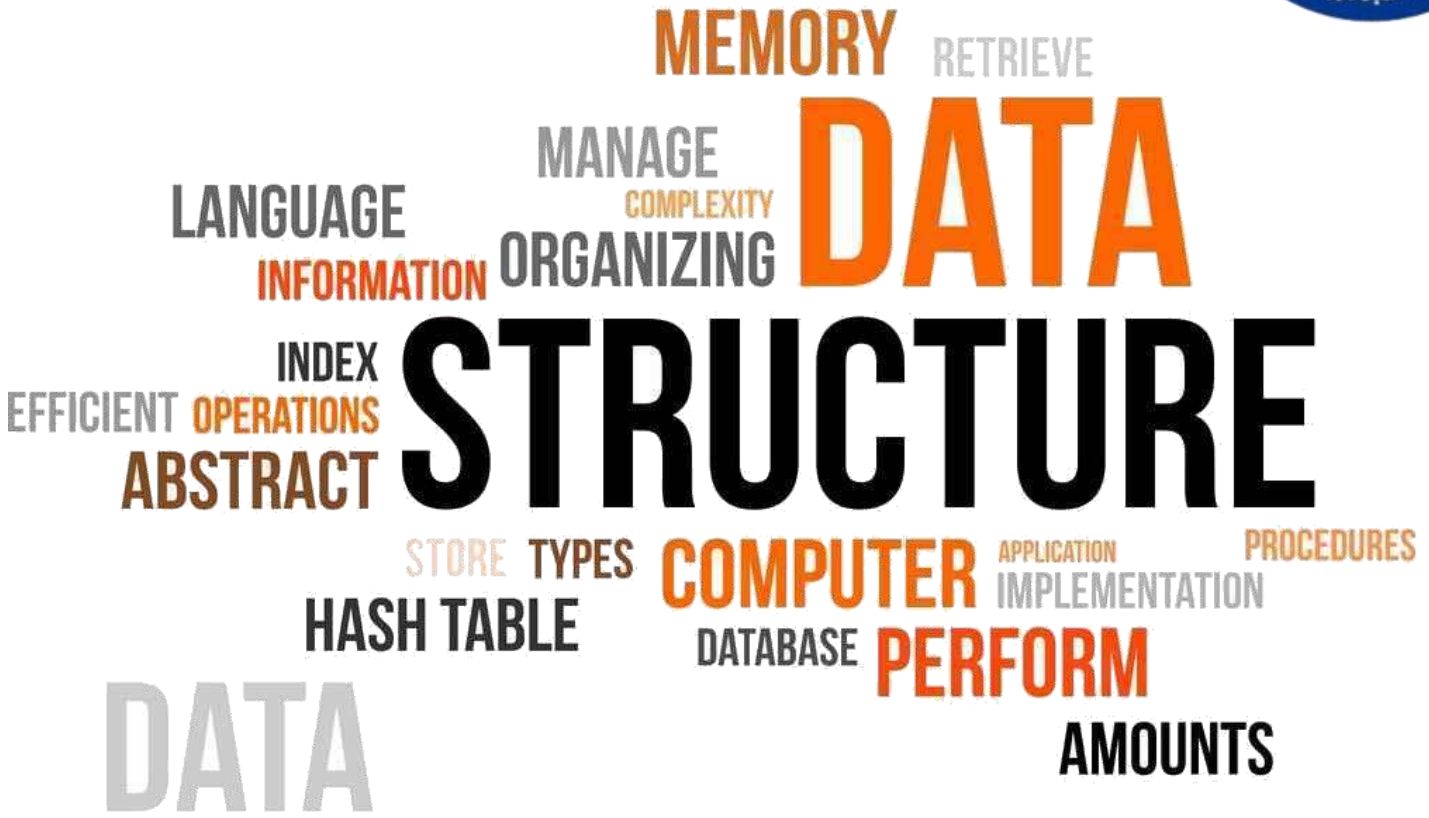


Data Structures

Course code: IT623



Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Linked List

* Representation of Linked List in Memory

⇒ Let LIST be a linked list.

⇒ LIST will be maintained in memory, unless specified on implied as follows.

1) LIST requires two linear arrays — INFO, LINK

Such that

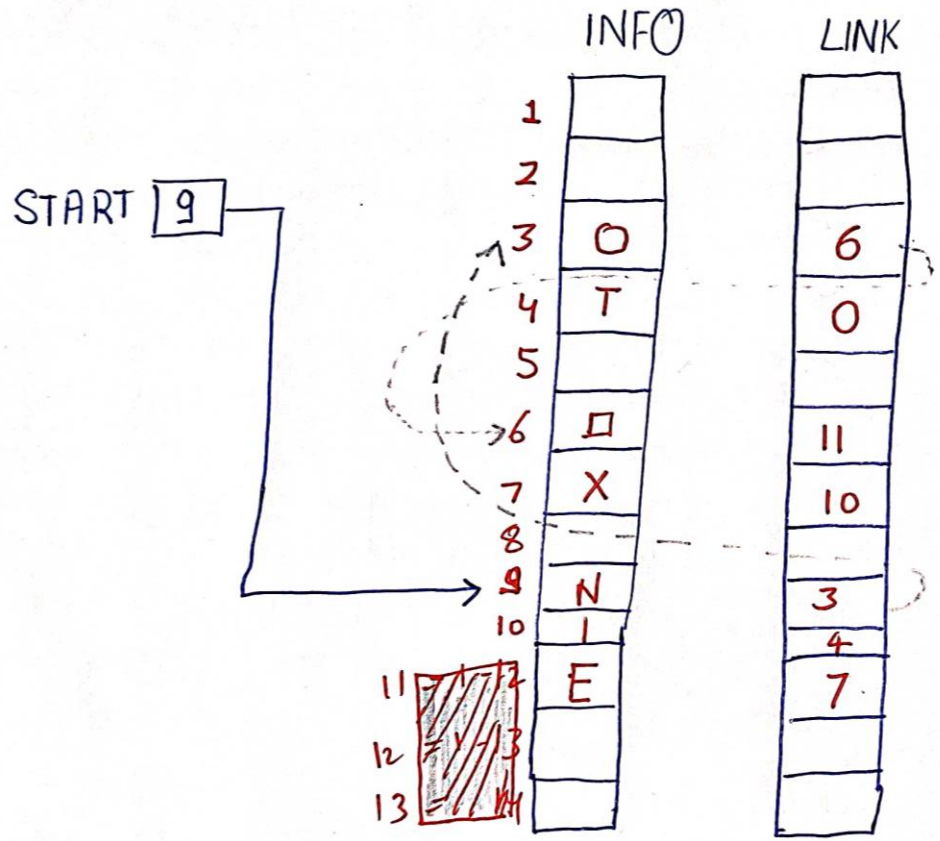
INFO[K] and LINK[K] contains the information and next pointer field, respectively.

2) LIST requires a variable name - "START" to store the location of the begining of the list, and a next pointer sentinal - denoted by NULL.

3) The subscripts of the arrays INFO and LINK will usually be positive, thus, we will choose NULL = 0, unless otherwise stated.

Example →

A linked list in memory where each node of the list contains a single character.



→ Two list can be maintained in the memory.

→ Where the nodes of both lists are stored in the same linear array.

↓

This was not possible in case of linear array or array data structure.

→ We see this in next example.

Example

The picture shows two lists of test scores, here **ALG** and **GEOM** may be maintained in memory where the nodes of both lists are stored in the same linear arrays **TEST** and **LINK**.

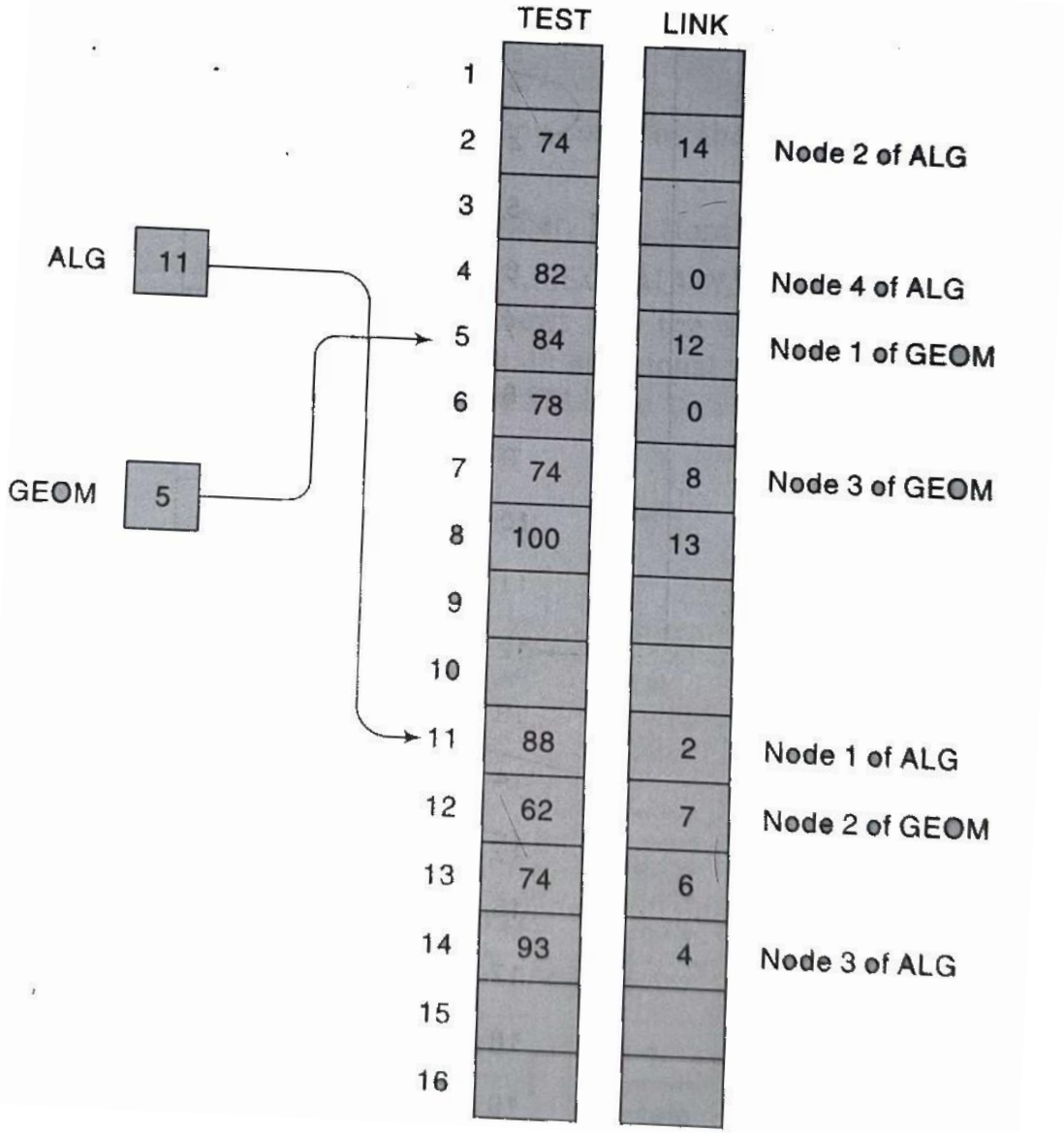
ALG contains **11**, the location of its first node, and GEOM contains **5,** the location of its first node.

ALG contains:

88, 74, 93, 82

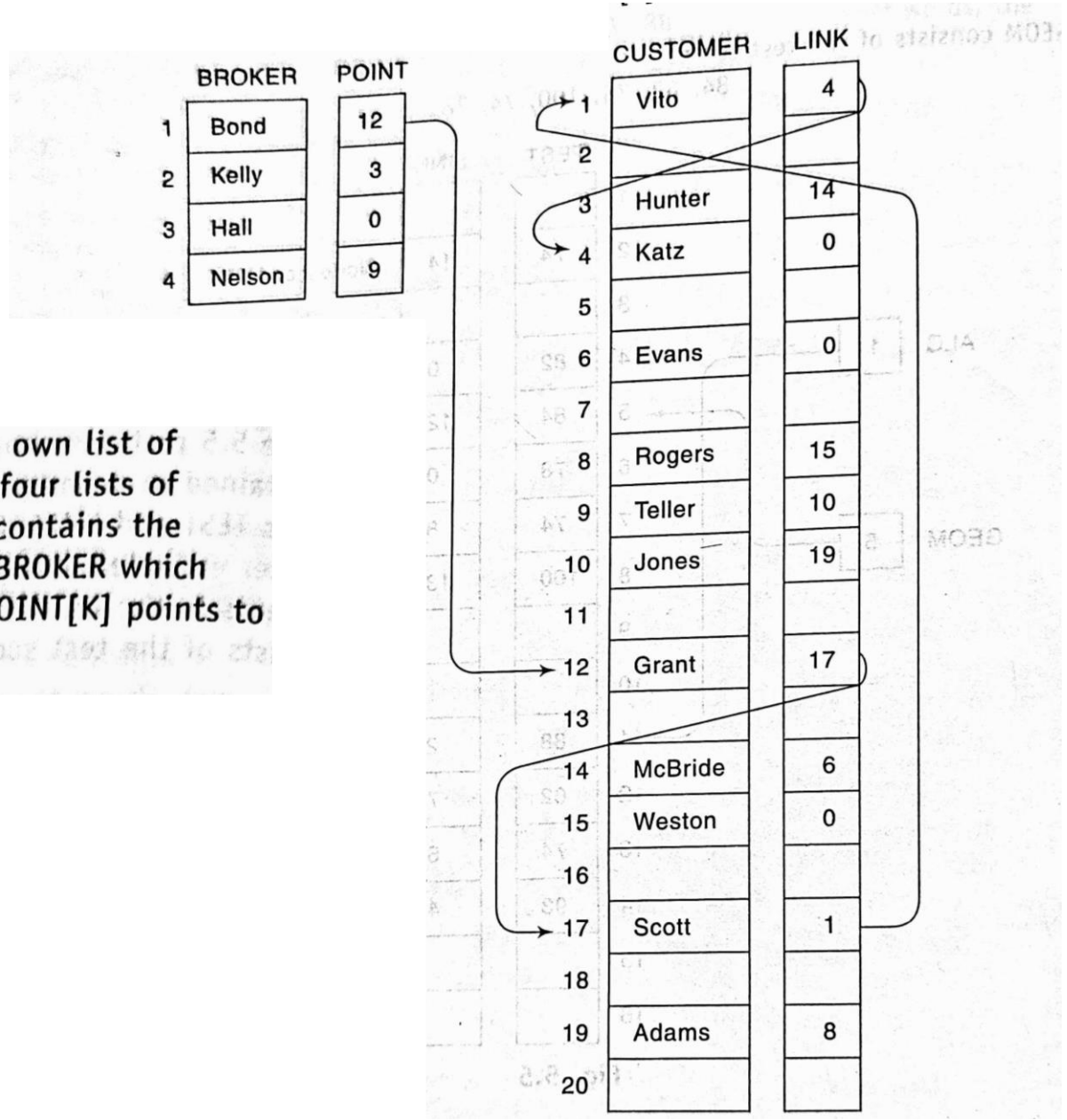
GEOM contains:

84, 62, 74, 100, 74, 78



Example

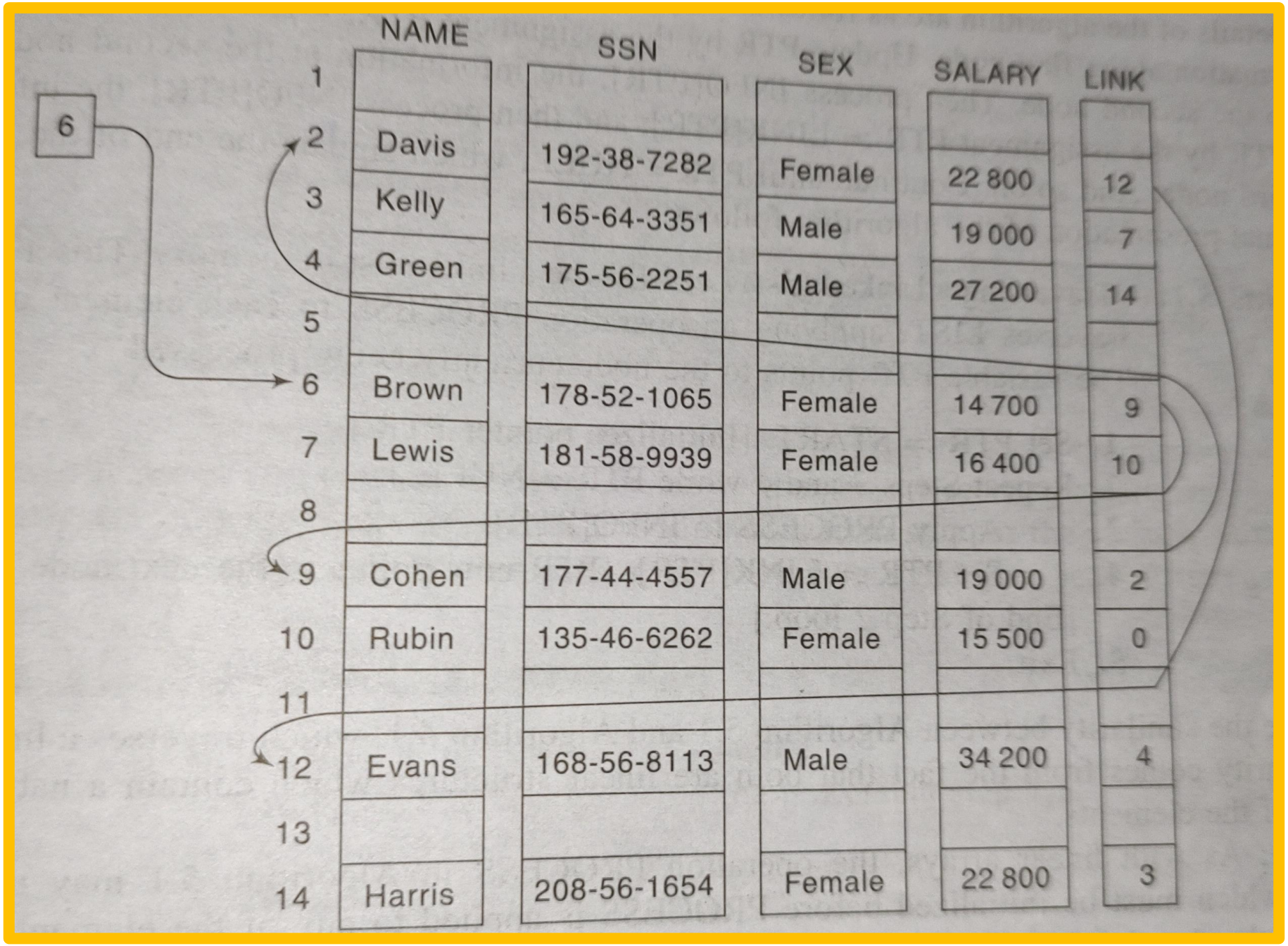
Suppose a brokerage firm has four brokers and each broker has his own list of customers. Such data may be organized as in Figure That is, all four lists of customers appear in the same array CUSTOMER, and an array LINK contains the nextpointer fields of the nodes of the lists. There is also an array BROKER which contains the list of brokers, and a pointer array POINT such that POINT[K] points to the beginning of the list of customers of BROKER[K].



Example

Name,
Social Security Number (SSN),…
Monthly Salary

Simultaneously addressing the issues of four or more linear arrays for adding different details.



Traversing a Linked List :

* { LIST, LINK, START, INFO } *

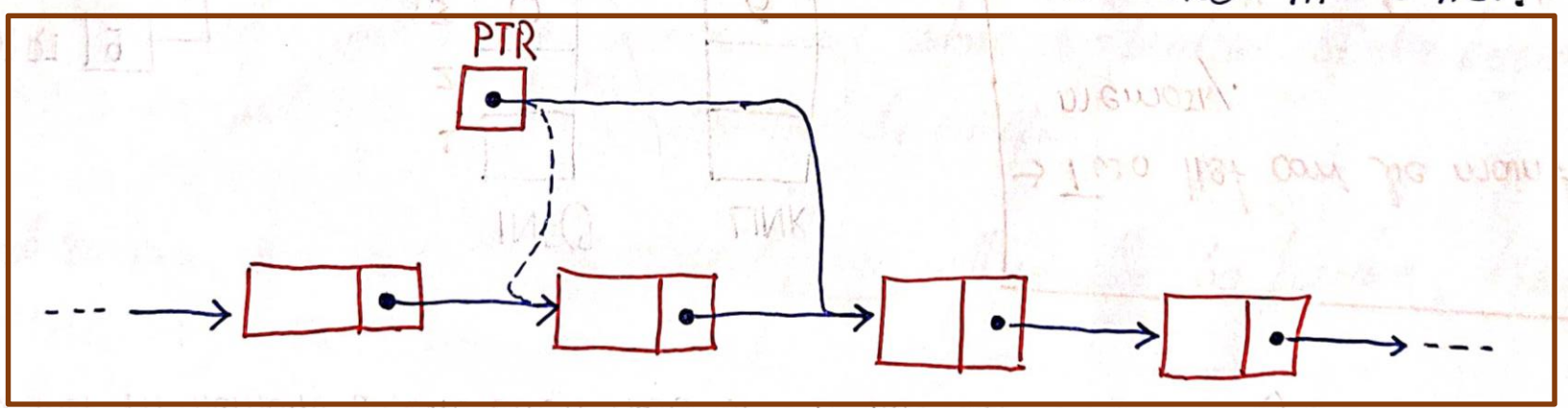
> Suppose we want to traverse LIST in order to process each node exactly once.

> The traversing algorithm uses a pointer variable PTR which points to the node that is currently being processed.

> LINK[PTR] points to the next node to be processed.

PTR := LINK[PTR]

moves the pointer to the next node in the list.



Traversing a Linked List

Algorithm: (Traversing a Linked List)

→ This algorithm traverses LIST, applying an operation PROCESS to each element of LIST.

→ The variable PTR points to the node currently being processed.

1. Set PTR = START [Initialize pointer PTR]
2. Repeat steps 3 and 4 while PTR ≠ NULL
3. Apply PROCESS to INFO[PTR]
4. Set PTR = LINK[PTR]
5. Exit.

Example of "PROCESS":

10-minutes Assignment

1) Print Information about each node ? /* Present element to be printed */

2) Count Number of element in each list?

3) Identify the second largest element?

(Hint: Use to initial variable;)

SEARCHING A LINKED LIST :-

* We assume the 'ITEM' of information is given.
* IF 'ITEM' is actually a key value and we are searching through a file for the record containing 'ITEM', then ITEM can appear only once in 'LIST'

Key value → Important one or identifier.

A.] LIST is Unsorted

* Let us consider the data in LIST are not necessarily "sorted."
* One searches for ITEM in LIST by traversing through the list using a pointer variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST.
*

```
PTR = LINK[PTR]   /* we update the pointer PTR */
```

* We require two tests

1) We have to check to see whether we have reached the end of the list:

PTR = NULL

2) IF not, then we check to see whether

INFO[PTR] = ITEM

* We use the first test to control the execution of a loop, and

* We let the second test take place inside the loop.

Algorithm → SEARCH (INFO, LINK, START, ITEM, LOC)

This algorithm finds the location (LOC) of the node where (ITEM) first appears in (LIST), or sets (LOC = NULL).

1. Set PTR = START
2. Repeat Step 3 while PTR ≠ NULL
3. if ITEM = INFO [PTR] then:
   Set LOC = PTR and Exit
   Else
   set PTR = LINK [PTR] /* PTR now points to the next node */
4. /* Search unsuccessful */ set LOC = NULL
5. Exit.