Basic Terminology

- **Data** are simply value or set of values.
- A data item refers to a single unit of values.
- Data items that are divided into sub items are called “**group items**”; those that are not divided are called “**elementary items**”. E.g.: name can be divided into sub items – first name, middle name & last name but the P.F.F. number is treated as a single item.
- Collections of data are organized into a hierarchy of fields, records and files.
- An **entity** is something that has certain attributes or properties, which may be assigned values. Example:
  
  Attributes: Name Age Sex PFN
  Values: Ruchi 21 F 1423

  - Entities with similar attributes (e.g. all the employees in an organization) form an entity set.
  - The term “**information**” is sometimes used for data with given attributes, or, in other words, meaningful or process data.
  - The way that data is organized into the hierarchy of fields, records & files reflects the relation between attributes, entities and entity sets.
  - That is, a **field** is elementary unit of information representing an attribute of an entity, a **record** is the collection of field values of a given entity & a **file** is the collection of records of the entities in a given entity set.

Q. **What is Data Structure? What are its different types?**

**Ans.** Data may be organized in many different ways; the logical or mathematical model of a particular organization of data is called a data structure.

There are many types of data structure. They are as follows:

1) **Arrays:**
The simplest type of structure is a linear (or one dimensional) array. By a linear array, we simply mean a list of finite number n of similar data element under same name. If we choose the name ‘A’ for the array, then the elements of ‘A’ are denoted by:
- subscript notation - ………………….. A₁, A₂,…………, Aₙ
- or by the parenthesis notation......A[1], A[2],………,A[n]
- or by the bracket notation…………..A(1), A(2),………,A(n)

  The number k in A[k] is called a **subscript** and A[k] is called a **subscripted** variable.

2) **Linked List:**
A linked list, or one-way list, is a linear collection of data elements, called nodes where the linear order is given by means of pointers. That is, each node is divided into two parts: the first part contains the information of the element, and the second part, called the link field or next pointer field, contains the address of the next node in the list.

3) **Record Structure:**
A file may be maintained by means of one or more arrays or records, where one indicates both the group items and the elementary items. It can also be described best by means of a tree
structure. For example, an employee personnel record may contain the following data items:
P.P.F. no, name, address, age and salary.
The record structure is presented as follows:

```
Employee
   /
  /   /
Name PFF no. Address Age Salary
   /     /
  FN MN LN City State Pin
```

Hierarchical representation Fig. (a)

Or
01. Employee
   02. Name
      03. FN
      03. MN
      03. LN
   02. PFF No
   02. Address
      03. City
      03. State
      03. Pin
   02. Age
   02. Salary

Level no. representation (Fig a)

A linear array is a list of finite number “n” of homogenous data elements (data elements of same
data) such that:

a) The elements of array are referred respectively by an index set consisting of “n” consecutive
   number.

b) The elements of array are stored respectively in successive memory locations.

Q. Explain One Dimensional Array. How it is represented in memory?
Ans. The number ‘n’ of elements is called the length or size of the array. The length or the number of
data elements of the arrays can be obtained from the index set by the formula.

\[
\text{Length} = UB - LB + 1
\]

where, UB is the length index called upper bound.
LB is the smallest index called lower bound.

- The following is an example of one dimensional array:-
  Let DATA be a 6-element array of integers such that DATA[1]=200, DATA[2] = 201,
  Simply writing as, DATA: 200,201,202,203,204,205
  The array DATA can be pictured as
Representation of linear arrays in memory:

Let “LA” be a linear array in the memory of computer. We know that the memory of the computer is simply a sequence of addressed locations as pictured below

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>200</td>
<td>201</td>
<td>202</td>
<td>203</td>
</tr>
<tr>
<td>201</td>
<td>1</td>
<td>201</td>
<td>202</td>
<td>203</td>
<td>204</td>
<td>205</td>
</tr>
<tr>
<td>202</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loc (LA[k]) = address of the element LA[k] of the array LA. As previously noted the elements of LA are stored in successive memory cells. Accordingly, the computer does not need to keep the track of the address of every element of LA, but needs of keep tracks only of the address of the first elements of LA, denoted by

Base (LA) i.e., starting address of array

Using this address Base (LA), the computer calculates the address of any elements of LA by the following formula:

\[ \text{LOC}(\text{LA}[k]) = \text{Base}(\text{LA}) + w(k - \text{lower bound}) \]

where w is the number of words per memory cells for the array LA.

**TRAVERSING LINEAR ARRAYS:** [Imp]

The following algorithm traverses a linear array LA. The simplicity of the algorithm comes from the fact that LA is a linear structure. Other linear structures, such as linked lists, can also be easily traversed. On the other hand, the traversal of nonlinear structure, such as trees and graphs, is considerably more complicated.

**Algo:** (Traversing a linear Array) here LA is a linear array with lower bound LB and upper bound UB. This algorithm traverses LA applying an operation PROCESS to each elements of LA.

1. Set K: = LB // Initialize counter
2. Repeat Steps 3 and 4 while K<UB
3. Apply PROCESS to LA[k]. // Visit element
4. Set K: = K+ 1. // Increase counter
   [End of Step 2 loop]
5. Exit.
We also state an alternative form of the algorithm, which uses a repeat-for loop instead of the repeat-while loop.

**Algo using ‘for’ loop:** (Traversing a Linear Array) this algorithm traverses a linear array LA with lower bound LB and upper bound UB.

1. Repeat for $K = LB$ to UB:
   - Apply PROCESS to LA[$k$]
   [End of loop]
2. Exit.

**Inserting and Deleting element:** [Imp]

Let A be a collection of data elements in the memory of the computer. “Inserting” refers to the operation of adding one more element to the collection A, and “deleting” refers to the operation of removing one of the elements from A. This section discusses inserting and deleting when A is a linear array.

Inserting an element at the “end” of the linear array can be easily done provide the memory space allocated for the array is large enough to accommodate the additional element. On the other hand, suppose we need to insert an element in the middle of the array. Then, on the average, half of the elements must be moved downward to new locations to accommodate the new element and keep the order of the other elements.

Similarly, deleting an element at the “end” of an array presents no difficulties, but deleting an element somewhere in the middle of the array would require that each subsequent element be moved one location upward in order to “fill up” the array.

**Algorithm:** (Inserting into a Linear Array)

**Alg 1:**

```
INSERT(LA, N,K,ITEM)
```

Here LA is a linear array with N elements and K is a positive integer such that $K \leq N$.

This algorithm inserts an elements ITEM into the $K^{th}$ position in LA.

1. Set $J = N$ //Initialize counter
2. Repeat Steps 3 and 4 while $J > K$.
3. Set LA
   
4. Set: $J = J - 1$. // Decrease counter
   
   [End of step 2 loop.]
5. Set LA $[k] := ITEM$. // Insert element
6. Set N := N+1 // Reset N
7. Exit.

The following algorithm deletes the $K^{th}$ element from a linear array LA and assigns it to a variable ITEM.
Algorithm: (Deleting from Linear Array)

DELETE (LA,N,K,ITEM)

Here LA is a linear array with N elements and K is a positive integer such that K<N. This algorithm deletes the K\textsuperscript{th} elements from LA.

1. Set ITEM: = LA[K].
2. Repeat for J = K to N-1:
   
   Set LA [J]:=LA[J+1]  
   Move J+1\textsuperscript{st} element upward
   [End of loop.]
3. Set N:=N-1  
   Reset the number N of elements in LA
4. Exit.

Q. Explain bubble sort algorithm with the help of example.

Ans. Let A be a list of n numbers. Sorting A refers to the operation of rearranging the elements of A so they are in increasing order. That is,


Example: 5,4,7,9,2,3; the sorted elements are 2,3,4,5,7,9

Bubble sort:

Suppose the list of numbers A[1], A[2], ..........A[N] is in memory. The bubble sort algorithm works as follows:


Note: During step 1, the largest element is “bubbled up” to the n\textsuperscript{th} position or “sinks” to the n\textsuperscript{th} position.

\textbf{Step2:} Repeat step 1 with one less comparison, that is, now we stop after we compare and possible rearrange A[N-2] and A[N-1].

\textbf{Step3:} Repeat step 1 with two fewer comparisons, that is, we stop after we compare and possible rearrange A[N-3] and A[N-2].

............................................................................................................................


The process of sequentially traversing through all or part of a list is frequently called a “pass” so each of the above step is called a pass.

\textbf{Example:}

Suppose the following numbers are stored in an array A

32,51,27,85,66,23,13,57

Now, we will apply bubble sort to the array
Pass1:
(a) Compare $A_1$ and $A_2$. Since $32 < 51$, the list is not altered.
(b) Compare $A_2$ and $A_3$. Since $51 > 27$, interchange the nos.
   \[32, 27, 51, 85, 66, 23, 13, 57\]
(c) Compare $A_3$ and $A_4$. Since $51 < 85$, the list is not altered.
(d) Compare $A_4$ and $A_5$. Since $85 > 66$, interchange the nos.
   \[32, 27, 51, 66, 85, 23, 13, 57\]
(e) Compare $A_5$ and $A_6$. Since $85 > 23$, interchange the nos.
   \[32, 27, 51, 66, 23, 85, 13, 57\]
(f) Compare $A_6$ and $A_7$. Since $85 > 13$, interchange the nos.
   \[32, 27, 51, 66, 23, 13, 85, 57\]
(g) Compare $A_7$ and $A_8$. Since $85 > 57$, interchange the nos.
   \[32, 27, 51, 66, 23, 13, 57, 85\]

At the end of pass 1, the largest number 85, has moved to the last position. However the rest of the numbers are not sorted, even though some of them have changed their positions.

Pass2:
\[27, 33, 51, 63, 23, 13, 57, 85\]
At the end of pass 2, the second largest number, 66, has moved to its way down next to the last position.

Pass3:
\[27, 33, 23, 51, 13, 57, 66, 85\]

Pass4:
\[27, 23, 33, 13, 51, 57, 66, 85\]

Pass5:
\[23, 27, 13, 33, 51, 57, 66, 85\]

Pass6:
\[13, 23, 27, 33, 51, 57, 66, 85\]

Pass7: Finally, there is no comparison now, as we have sorted the elements of the array $A$.

**Algorithm: (imp)**

1. Repeat step 2 and 3 for $K = 1$ to $N-1$.
2. Set $PTR:=1$ \hspace{1cm} //Initialize pass pointer $PTR$
3. Repeat while $PTR \leq N-K$ \hspace{1cm} //Execute pass
a) If DATA [PTR] > DATA [PTR+1], then
   Interchange DATA [PTR] and DATA [PTR+1]
   [End of If structure]
b) Set PTR := PTR+1.
   [End of inner loop.]
   [End of step 1 outer loop]

4. Exit.

There is an inner loop, which is controlled by the variable PTR and the loop is contained in an outer loop, which is controlled by an index K.

Q. Explain binary Search algorithm with the help of example.

Ans. Algorithm:

Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID, respectively, denotes the beginning, end and middle locations of a segment of element of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

1. [Initialize segment variables.]
   Set BEG = LB, END = UB and MID = INT(BEG + END/2)
2. Repeat step 3 & 4 [while BEG, END and DATA [MID], ITEM]
3. If ITEM < DATA [MID], then:
   Set END = MID - 1.
   Else:
   Set BEG = MID + 1.
   [End of If structure]
4. Set MID = INT((BEG + END)/2)
   [End of step 2 loop]
5. If DATA [MID] = ITEM, then:
   Set LOC = MID
   Else:
   Set LOC = NULL
   [End of If structure]

Example:

Let DATA be the following sorted 13-element array:
DATA: 11,22,30,33,40,44,55,60,66,77,80,88,99
We apply the binary search to DATA for different values of ITEM.

Suppose we are searching ITEM = 40
So, it will be like
Initially, BEG = 1 and END = 13. Hence
MID = INT((1 + 13)/2) = 7 & so DATA[MID] = 55
Since $40 < 55$, END has its value changed by $\text{END} = \text{MID} - 1 = 6$. Hence, 
$\text{MID} = \text{INT}[(1+6)/2] = 3$ & so $\text{DATA}[\text{MID}] = 30$

Since $40 > 30$, BEG has its value changed by $\text{BEG} = \text{MID} + 1 = 4$ Hence 
$\text{MID} = \text{INT}[(4+6)/2] = 5$ & so $\text{DATA}[\text{MID}] = 40$

We have found ITEM in location $\text{LOC} = \text{MID} = 5$.

1) 11 22 30 33 40 44 55 60 66 77 80 88 99
2) 11 22 30 33 40 44 55 60 66 77 80 88 99
3) 11 22 30 33 40 44 55 60 66 77 80 88 99

[Successful ]

Binary Search for ITEM = 40.

Q) Write an algorithm for linear search technique with suitable example.

Ans. Algorithm : Linear Search

\text{LINEAR (DATA, N, ITEMS, LOC)}

Here DATA is a linear array with N elements and ITEM is given element. This algorithm finds the location LOC of ITEM in DATA or sets LOC = 0, if search is successful.

Step 1 : [Insert ITEM at the end of DATA]
Set $\text{DATA}[N+1] = \text{ITEM}$

Step 2 : [Initialize counter]
Set LOC = 1

Step 3 : [Search for item]
Repeat While $\text{DATA}[\text{LOC}] \neq \text{ITEM}$ :
Set LOC = LOC + 1
[End of loop]

Step 4 : If LOC = N + 1, then :
Set LOC = 0

Step 5 : Exit

For example : Given DATA array with following 5 elements
11 22 33 44 55
Suppose ITEM = 33

Step 1 : Set $\text{DATA}[6] = 33$, $\therefore$ List becomes
11 22 33 44 55 33

Step 2 : LOC = 1

Step 3 : Since DATA [1] = 11 \neq 33 $\therefore$ LOC = 2
Since DATA [2] = 22 \neq 33 $\therefore$ LOC = 3
Here DATA [3] = 33 = 33 = ITEM

Step 4 : Hence ITEM = 33 found at position, LOC = 3.
Q) Write difference between Linear search and Binary search.

Ans.

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Linear search performs on unsorted list of elements as well as sorted list.</td>
<td>1. For binary search, the elements in array are stored in alphabetically or numerically in sorted manner.</td>
</tr>
<tr>
<td>2. Compare the desired element with all elements in an array until the match is found.</td>
<td>2. Compare the value of midpoint with desired value. If the value is greater than midpoint value the first half is checked, otherwise second half is checked until search is successful or interval is empty.</td>
</tr>
<tr>
<td>3. Insertion of an element in an array can be performed very efficiently when array is not ordered.</td>
<td>3. An insertion of a new element requires that many elements be physically moved to preserved order.</td>
</tr>
<tr>
<td>4. For large size of array, time required for this search is very large.</td>
<td>4. For large size of array, comparatively time required is less.</td>
</tr>
<tr>
<td>5. Time complexity is as follows: worst case : N comparison Best case : 1 comparison</td>
<td>5. Time complexity as follows: worst case : $\log^2 N$ comparison Best case : 1 comparison</td>
</tr>
</tbody>
</table>
Arrays, Records and Pointers

INTRODUCTION:
Data structures are classified as either linear data structure or non-linear data structure.

- **Linear data structure**-
  A data structure is said to be linear if its elements form a sequence, or in other words, a linear list. There are two basic ways of representing such linear structure in memory.
  1) The first way is to have the linear relationship between the elements of sequential memory locations. These linear structures are called as arrays.
  2) The second way is to have the linear relationship between the elements represented by means of pointers or links. These linear structure are called linked lists.

- **Non-linear data structure**-
  This structure is mainly used to represent data containing a hierarchical relationship between elements, that is, records, family trees and table of contents such as trees and graphs.

**Linear arrays:**
**Definition:** An array can be defined as the collection of the sequential memory locations, which can be referred to a single name along with a number, known as index, to access a particular field or data. When the elements need to be referred by more than one index, it is known as Multidimensional arrays.

There are data structures other than arrays, linked lists and trees. Some of these structures are as follows:

a) **Stack:** A stack, also called a last-in-first-out (LIFO) system, is a linear list in which insertion and deletion can take place only at one end, called the top. This structure is similar in its operation to a stack of dishes or glasses placed at one above another and you can take out the dishes or glass only from one end, that is, top.

b) **Queue:** A queue, also called a first-in-first-out (FIFO) system, is a linear list in which deletion can take place only at one end of the list “front”, and insertions can take place only at the other end of the list “rear”. For example, Queue in the bus stop, the first person in the queue gets in the bus first and to join the queue the person has to come in the last.

c) **Graph:** Data sometimes contain a relationship between pairs of elements, which is not hierarchical in nature. For Example, suppose an airline flies only between the cities connected by lines. The data structure that reflects this type of relationship is called a graph.

Q. **What are data structure operations?**
**Ans.** The data appearing in our data structures are processed by means of certain operations. The following are the data structure operations, which are used:

1. **Traversing:** Accessing each record exactly only once so that certain items in the record may be processed. (This accessing & processing is sometimes called “visiting” the record)
2. **Searching**: Finding the location of the record with the given key value, or finding the locations of all the records.
3. **Inserting**: Adding a new record to the structure.
4. **Deleting**: Removing a record from the structure.
5. **Sorting**: Arranging the records in some logical order (e.g. alphabetically or numerically).
6. **Merging**: Combining the records in two different sorted files into a single sorted file operations.

Q. **What are the different control structures?**

**Ans.** The control structures are as follows:

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

**Explanation:**

1. **Sequential flow:**
   
   It means that the flow of the structure flows in sequence. The sequence can be presented by means of numbered steps or by which the modules are written. Example:

   ![Sequential Flow Diagram]

   *Fig (a) Sequential flow*

2. **Selection logic**: (Conditional Flow):
   
   Selection logic employs a number of conditions, which lead to a selection of one out of several alternative modules. The structures which implement this logic are called conditional structures or IF structures. These conditional structures fall into three types. They are
   
   (a) **Single alternative**: This structure has the form:
       
       If condition, then:
       
       [Module A]
       
       [End of If structure]
       
       The logic of this structure is pictured as:

   ![Single Alternative Diagram]

   *Single alternative*

   (b) **Double alternative**: This structure has the form:
       
       If condition, then:
       
       [Module A]
       
       [Module B]
       
       The logic of this structure is pictured as:

   ![Double Alternative Diagram]

   *Double alternative*
(b) Double alternative: This structure has the form

IF condition, then:
[Module A]
Else:
[Module B]
[End of If structure]

The logic of this structure is pictured in Fig (b), above.

(c) Multiple alternative: This structure has the form

If condition (1), then:
[Module A]
Else if condition (2), then:
[Module B]
Else:
[Module C]
[End of If structure]

This logic of this structure allows only one of the modules to be executed.

3. **Iteration Logic** (Repetitive Flow):

This type of logic refers with a repeat statement involving loops. Each type begins with a Repeat statement and is followed by a module, called the *body of the loop*

Q. **What is the Complexity of Algorithms?**

Ans.
- An algorithm is a well-defined list of steps for solving a particular task or problem.
- One major purpose of complexity is to develop efficient algorithm for the processing of or data.
- The *time* and *space* it uses are two major measures of the efficiency of an algorithm.
  - The complexity of an algorithm is the function which gives the running time / space in terms of the input size.
- The following two complexities are the important factor for complexity of an algorithm.
• Space complexity
• Time complexity

**Definition:**

(i) **Space complexity:** The space complexity of an algorithm is the amount of memory it needs to run or to complete an algorithm.

(ii) **Time complexity:** The time complexity of an algorithm is the amount of computer time it needs to run or to complete an algorithm.

**Q. What is a record? How it differs from a linear array?**

**Ans.** A record is a collection of field or attributes i.e. relative data items.

Collection of data is frequently organized into hierarchy of field i.e. records. A file is nothing but collection of records.

Difference between records and linear arrays:

(i) A record is a collection of fields, while an array is list of homogeneous data elements.

(ii) A record may contain non-homogeneous data i.e. data elements may be of different data types. An array always contains homogeneous data.

(iii) In a record, natural ordering of elements is not possible. Array elements can be naturally ordered.

(iv) Elements of record are referenced by level number, while those of array can be referenced by an index set consisting of n consecutive numbers.
Linked List

ARRAY

- Data processing involves storing and processing data organized into lists.
- One-way store such data is by means of arrays.
- The linear relationship between the data elements of an array is reflected by the physical relationship of
  the data in memory not by any information contained in the data elements themselves.
- This makes easy to compute the address of an element in an array.
- **Disadvantages:** It is relatively expensive to insert and delete elements in an array.
- Also, since an array usually occupies a block of memory, one cannot simply double or triple the size of
  the array.

Link or Pointer

- Another way of storing a list in memory is to have each element in the list contain a field, called list or
  pointer, which contains the address of the next element in the list.
- It does not occupy the adjacent space in memory, this will make it easier to insert and delete
  elements in the list.

**Q.** What is Linked list? Explain.

**Ans.**

A linked list, or *one-way list*, is a linear collection of data elements, called *nodes* where the linear order
is given by means of pointers. That is, each node is divided into two parts: the first part contains the
information of the element, and the second part, called the link field or *next pointer field*, contains the
address of the next node in the list.

**Basic example of linked list:**

The diagram below of a linked list contains 6 nodes. Each node contains two parts. The left part
represents the information of the present node, which may contain an entire record of data items
(e.g. NAME, ADDRESS......). The right part represents the next pointer field of the node, and
there is an arrow drawn from it to the next node in the list. The pointer of the last node contains
a special value, called the *null* pointer, which is any invalid address.

**Start**

![Diagram of linked list]

**Next pointer field of fourth record**

**Information of third record**

**Example:**

A hospital ward contains 12 beds, of which 9 are occupied. Suppose we, arrange as listing of the
patients. This list can be given in the pointer field, Called NEXT. We use the variable START
to point to the first patient. Hence START contains 5.5 contain the information Vishal & points
to bed 3. Since 3 contain the patient Karan & Points to bed 11. Since 11 contain the patient Sahil & points to bed 8 and so on.

<table>
<thead>
<tr>
<th>Bed. no.</th>
<th>Patient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sita</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Karan</td>
</tr>
<tr>
<td>4</td>
<td>Juliet</td>
</tr>
<tr>
<td>5</td>
<td>Vishal</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Ruchi</td>
</tr>
<tr>
<td>8</td>
<td>Mihir</td>
</tr>
<tr>
<td>9</td>
<td>Nita</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Sahil</td>
</tr>
<tr>
<td>12</td>
<td>Nilesh</td>
</tr>
</tbody>
</table>

Q. How linked list is represented in memory?

Ans.

Let LIST be a linked list. LIST requires two linear arrays-called as INFO and LINK (INFO contain the information of the present node and LINK contain the next pointer field of a node). LIST also requires a variable name- such as NAME or START, which contain the location of the beginning of the node and a NULL to indicate the end of the node.

The following example of linked list indicaes that the nodes of a list need not occupy adjacent elements in the arrays INFO and LINK. However, each node has its own pointer variable. From the given example, we can obtain the actual list of characters, or in other words the string.
Q. What are linked lists? Show a liked list with suitable example having six nodes with a properly labelled diagram.

OR

With suitable example, show labelled diagram for link between two nodes having the information part and next pointer field.

Ans. i) A linked list is a linear collection of data elements, called nodes, where the linear order is maintained with the help of pointers.

ii) Linked list is also called as one-way list.

iii) Each node in the linked list is divided into two parts. First part is called as INFO part, called as LINK part, which is next pointers field i.e., it contains the address of next node in the list.

iv) e.g.

(a) The above figure shows a linked list having six nodes.

(b) The left part of the node is the Info part, which contains information of the element, while the right part is Link part, which is next pointers field i.e., it points to next node.
(c) An arrow is drawn from Link part of one node to the next node to indicate link.
(d) The address of the list is stored in Start or Name of the list.
(e) The Link part of last node is null pointer i.e., it contains nothing.
(f) To trace the linked list, we just require the address of Start or Name.

**Binary Tree**

- We know the linear types of data structures: strings, arrays, lists, stacks and queues.
- Data structures also define nonlinear data structure called a tree.
- The tree structure is mainly used to represent data containing a hierarchical relationship between elements: example: family trees, records etc.
- **Binary tree** is a tree, which can be easily maintained in the computer.

**Q. What is Binary tree?**

**Ans.** A binary tree $T$ is defined as a finite set of elements, called nodes, such that:
(a) $T$ is empty (called the null tree or empty tree), or
(b) $T$ contains a distinguished node $R$, called the root of $T$ and the remaining nodes of $T$ form an ordered pair of binary trees $T_1$ and $T_2$.
(c) If $T$ is non-empty, then its root is called the left successor of $R$; similarly if $T$ is non-empty, then its root is called the right successor of $R$.

A binary tree $T$ is frequently presented by means of a diagram. Specifically, the diagram below represents a binary tree as follows:

(i) $T$ consists of 11 nodes
(ii) The root of $T$ is the node A.
(iii) A left-downward slanted line from a node $N$ indicates a left successor of $N$, and a right-downward slanted line from $N$ indicates right successor of $N.$

Observe the following tree:-

```
        A
       / \  \
      B   C
     / \  /  \
    D  E F   G
   / \    /  \
  H  I   J   K
```
Any node \( N \) of a binary tree \( T \) has either 0, 1, or 2 successors. The nodes A, B, C, G has two successors while the nodes E, I have only one successor and the nodes H, K have no successors. The nodes with no successors are called \textit{terminal nodes}.

The above definition of binary tree \( T \) is recursive since \( T \) is defined in terms of binary tree \( T \) and \( T_1 \). This means, that every node \( N \) of \( T \) contains a left and a right subtree. Moreover, if \( N \) is a terminal node, then both its left and right subtrees are empty.

Binary trees \( T \) and \( T_1 \) are said to be similar if they have same structure or shape. The trees are said to be copies if they are similar and if they have same contents at corresponding nodes.

\textbf{Example of Binary Tree:}

Consider any algebraic expression \( E \) involving only binary operations, such as
\[
E = (a - b) / ((c * d) + e)
\]
The binary tree will be constructed as:

![Binary Tree Diagram]

\textbf{Q. How binary tree is represented in memory?}

\textbf{Ans.} A tree \( T \) can be maintained in memory be means of a linked representation which uses three parallel arrays \( \text{INFO} \), \( \text{LEFT} \) and \( \text{RIGHT} \) and a pointer variable \( \text{ROOT} \). Each node \( N \) of \( T \) will correspond to a location \( K \) such that:

1) \( \text{INFO}[K] \) contains the data at the node \( N \).
2) \( \text{LEFT}[K] \) contains the location of the left child of node \( N \).
3) \( \text{RIGHT}[K] \) contains the location of the right child of node \( N \).
Example:-

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>NAME</th>
<th>LEFT SUCCESS OR</th>
<th>RIGHT SUCCESS OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Sita</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Guru</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Girish</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Bhawna</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Leeta</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Chandu</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>Ruby</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Amit</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Heena</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

a) The value ROOT = 14 indicates that Heena is the root of the tree. LEFT[14]=9 indicates that Chandu is the left child of Heena, and RIGHT[14]=7 indicates that Leeta is the right child of Heena. (Repeat the step for each node in the diagram)

Q. How to traverse binary tree?

Ans. There are three standard ways of traversing a binary tree T with root R. These three algorithms, called preorder, inorder and postorder, are as follows:

**Preorder:** (NLR)
1. Process the root R.
2. Traverse the left subtree of R in preorder.
3. Traverse the right subtree of R in preorder.
**Inorder:** (LNR)
1. Traverse the left subtree of R in inorder.
2. Process the root R.
3. Traverse the right subtree of T in inorder.

**Postorder:** (LRN)
1. Traverse the left subtree of R in postorder.
2. Traverse the right subtree of R in postorder.
3. Process the root R.

The three algorithms are sometimes called, respectively, the node-left-right (NLR) traversal, the left-node-right (LNR) traversal and the left-right-node(LRN) traversal.

**Q. Explain LIFO and FIFO.**  
**Or**  
**Explain Stack & Queue.**

**Ans.** A stack is a linear data structure in which the data elements can be inserted and deleted at one end only i.e. the top. LIFO stands for Last-in-First-out. The element which is inserted last in stack will be removed (deleted) first. The everyday example of Stack is stack of dishes, stack of plastic cups.

When you want to insert any element in stack, it is known as “PUSH” and when you want to delete any element from stack, it is known as “POP”.

**Example:** Suppose 11,12,43,24 ; is a set of data elements

Inserting element 65

![Diagram of pushing 65 onto a stack](https://via.placeholder.com/150)

Suppose if we want to delete 43 from the stack, so we have to first delete the top element i.e. 65, then the second, then third, until we pop the required element.

![Diagram of popping elements from a stack](https://via.placeholder.com/150)
Now, the stack contains the elements 11, 12.

FIFO
A queue is a linear data structure in which data elements can be inserted from one end and deleted from the other end. The element which is inserted first is deleted first and the element which has entered in the end will be deleted last, so the term FIFO. The everyday example of queue is line in a bus stop. The person standing first in the queue enters first in the bus and the person who has entered last in the queue, gets in last in the bus.

We insert the element from the “rear” and delete the element from the “front”.

Example: Suppose, 11,12,43,24 is a set of data elements

![Queue Diagram]

Inserting element 65.

![Queue Diagram with 65]

Now, if we want to delete the element 11.

![Queue Diagram without 11]

Q. What are pointer arrays?
Ans. i) An array is called pointer array, if each element of that array is a pointer.
ii) The variable is called as pointer variable, it points to another variable i.e. it contains the memory address of other variable.
iii) consider an organization, with divides its employee list into four groups, depending on certain conditions. Following figure shows the list of 4 groups.

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
<th>Group 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deepak</td>
<td>Swapnil</td>
<td>Rajdeep</td>
<td>Yashwant</td>
</tr>
<tr>
<td>Nitin</td>
<td>Amit</td>
<td>Amol</td>
<td>Chintamani</td>
</tr>
<tr>
<td>–</td>
<td>Vivek</td>
<td>Yogesh</td>
<td>Kishore</td>
</tr>
<tr>
<td>–</td>
<td>Ravi</td>
<td>Shekhar</td>
<td>Rohit</td>
</tr>
<tr>
<td>–</td>
<td>Omprakash</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
iv) If these groups are to be represented in memory, the most efficient way is to use 2 arrays. The first is Employee array, which contains list of employees in all four groups sequentially, while the second array is Group array, which is a pointer array, which contains the starting address of each group in the Employee array, respectively.

v) It is shown in figure:

![Employee and Group Arrays Diagram]

vi) Each element of Group array is a pointer, which holds the starting addresses of different groups. Hence, it is called as pointer array.

Q. **Draw the binary tree for the following expression:**

1. \( E = (a + b) / [(c * d) - e] \)
2. \((2x + y) (a - 7b)^3\)