GLS UNIVERSITY

0301302 DATA STRUCTURES.
UNIT– I

- Prof. Nirav Suthar
- Prof. Ankita Shah

INTRODUCTION TO DATA STRUCTURE

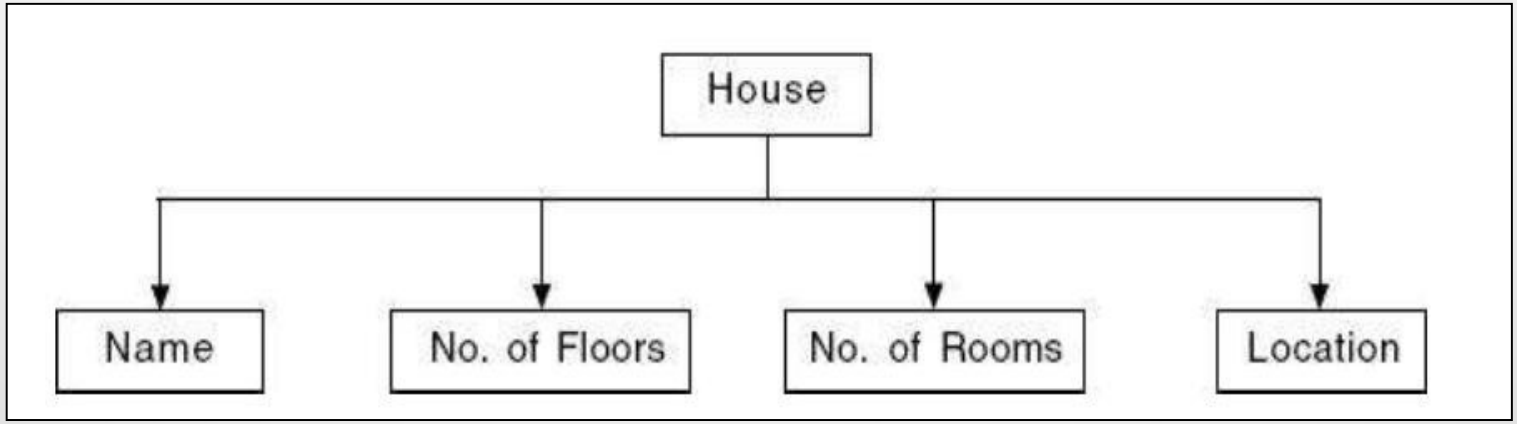
- **DATA:** Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way.
- Data is nothing but a piece of information.
- **For example :**
  - player's name : "Virat"
  - age : 26.
  - Here "Virat" => String data type
  - 26 => integer data type.

- **INFORMATION:** Meaningful data or processed data. Information is used for data with its attributes.
- **Example**:

Data	**Meaning**
22	Age of person
Herry	Nick name of person
22/4/15	DOB of person

PROBLEM ANALYSIS

- Data structures are a method of representing of logical relationships between individual data elements related to the solution of a given problem.
- Data structures are the most convenient way to handle data of different types.
- For example, the characteristics of a house can be represented by the house name, house number, location, number of floors, number of rooms on each floor, kind of fencing to the house—either with brick walls or wire, electrification—either underground or open, whether a balcony has been provided or not, etc.



PROBLEM ANALYSIS

- Generally, we define a PROBLEM a single statement that can be define in some general terms.
  - e.g. what is an output of the program??
  - write a program to traverse an elements in an array???
- **ANALYSIS USING AN ALGORITHM:** An algorithm is well defined steps for solving the computational problem in order to get better eficiency and a analysing of the PROBLEM.

Problem Analysis Chart (PAC)

separates the problem in 4 parts

Given Data	**Required Results**
Section 1:Data given in the problem or provided by user-Data, constants, variables	Section 2:Requirements to produce the output-information and format required
Processing required	**Solution alternatives**
Section 3:List of processing required – equations, or searching or sorting techniques	Section 4:List of ideas for the solution.

PROBLEM ANALYSIS

TASK :- 1) create a Problem analysis chart for the average problem

2) create a Problem Analysis chart for calculating the Gross pay , given the formula GrossPay = Hours * PayRate

Given Data	Required Results
Hours Pay Rate	Gross Pay
Processing required	**Solution alternatives**
GrossPay = Hours * PayRate	1.Define the hours worked And pay rate as constants 2. Define the hours worked and pay rate as input values

ALGORITHM ANALYSIS

- The analysis of algorithms is the determination of the amount of resources (such as time and storage) necessary to execute them. Most algorithms are designed to work with inputs.
- Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation, as mentioned below –

- **A priori analysis –** This is theoretical analysis of an algorithm. Efficiency of algorithm is measured by assuming that all other factors e.g. processor speed, are constant and have no effect on implementation.

- **A posterior analysis –** This is empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

- The efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps **(time complexity) or storage locations (space complexity).**

ALGORITHM ANALYSIS: COMPLEXITY

- There can be several ways to organize data or write algorithm for a given problem.
- We can compare one algorithm with the other and choose the best.
- Analysis involves measuring the performance of an algorithm.
- **COMPLEXITY:**
  - **Algorithms are measured in terms of time and space complexity.**
  - **Time complexity of an algorithm is a measure of how much time is required to execute an algorithm for a given number of inputs and is measured by its rate of growth.**
  - **Space complexity of an algorithm is a measure of how much storage is required by the algorithm.**

SPACE & TIME COMPLEXITY

- **Space complexity** is the amount of computer memory required during program execution aw a function of input size.
- Space complexity is measured in two ways:
  - **Compile time**
  - **Run-time**
- Compile time space complexity is defined as the storage requirements ofa program at compile time.
- If the program is recursive or uses dynamic variables then there is need to determine space complexity at run-time.

- Time Complexity is the time taken by a program that is sum of its compile and execution times.

BEST, WORST & AVERAGE CASES

- **The Best Case Complexity** of an algorithm is the function defined by the minimum number of steps taken on any instances of size n.

- **The Worst Case Complexity** of an algorithm is the function defined by the maximum number of steps taken on any instances of size n.

- **The Average Case Complexity** of an algorithm is the function defined by an average number of steps taken on any instances of size n.

Primitive and Non-Primitive data Types

- **The primitive data types** are the basic data types that are available in most of the programming languages. The primitive data types are used to represent single values. Programmers can use these data types when creating variables in their programs.
- Primitive data types are predefined types of data.

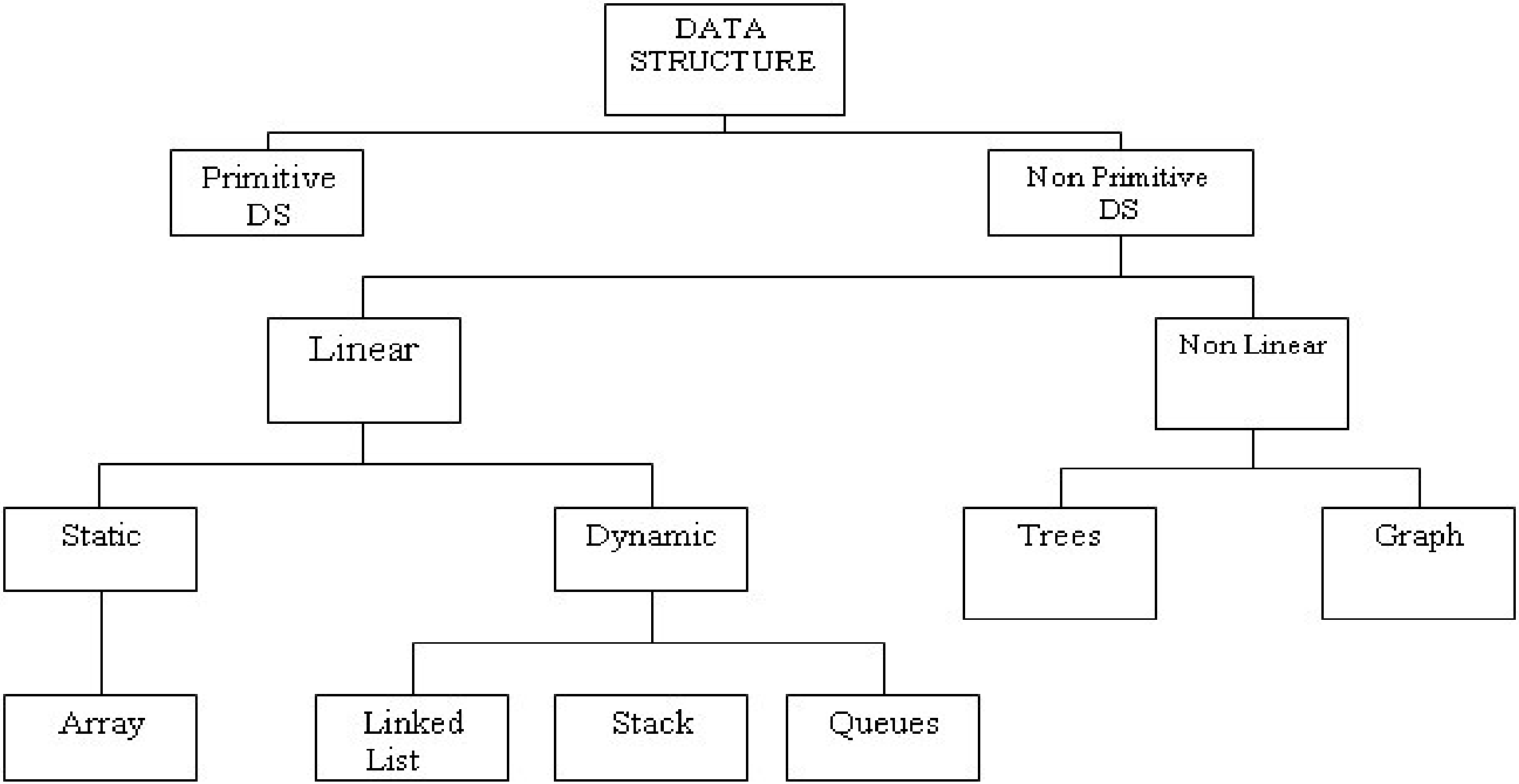
- **Integer**: This is used to represent a number without decimal point.
  - Eg: 12, 90
- **Float and Double**: This is used to represent a number with decimal point.
  - Eg: 45.1, 67.3
- **Character** : This is used to represent single character
  - Eg: ‘C’, ‘a’
- **String**: This is used to represent group of characters.
  - Eg: "M.S.P.V.L Polytechnic College"
- **Boolean**: This is used represent logical values either true or false.

Primitive and Non-Primitive data Types

- **NON-PRIMITIVE DATATYPES:** The data types that are derived from primary data types are known as non-Primitive data types. These datatypes are used to store group of values.
- Non-primitive data types are not defined by the programming language, but are instead created by the programmer.

- The non-primitive data types are:
  - Arrays
  - Structure
  - Union
  - linked list
  - Stacks
  - Queue etc

Types of Data Structures

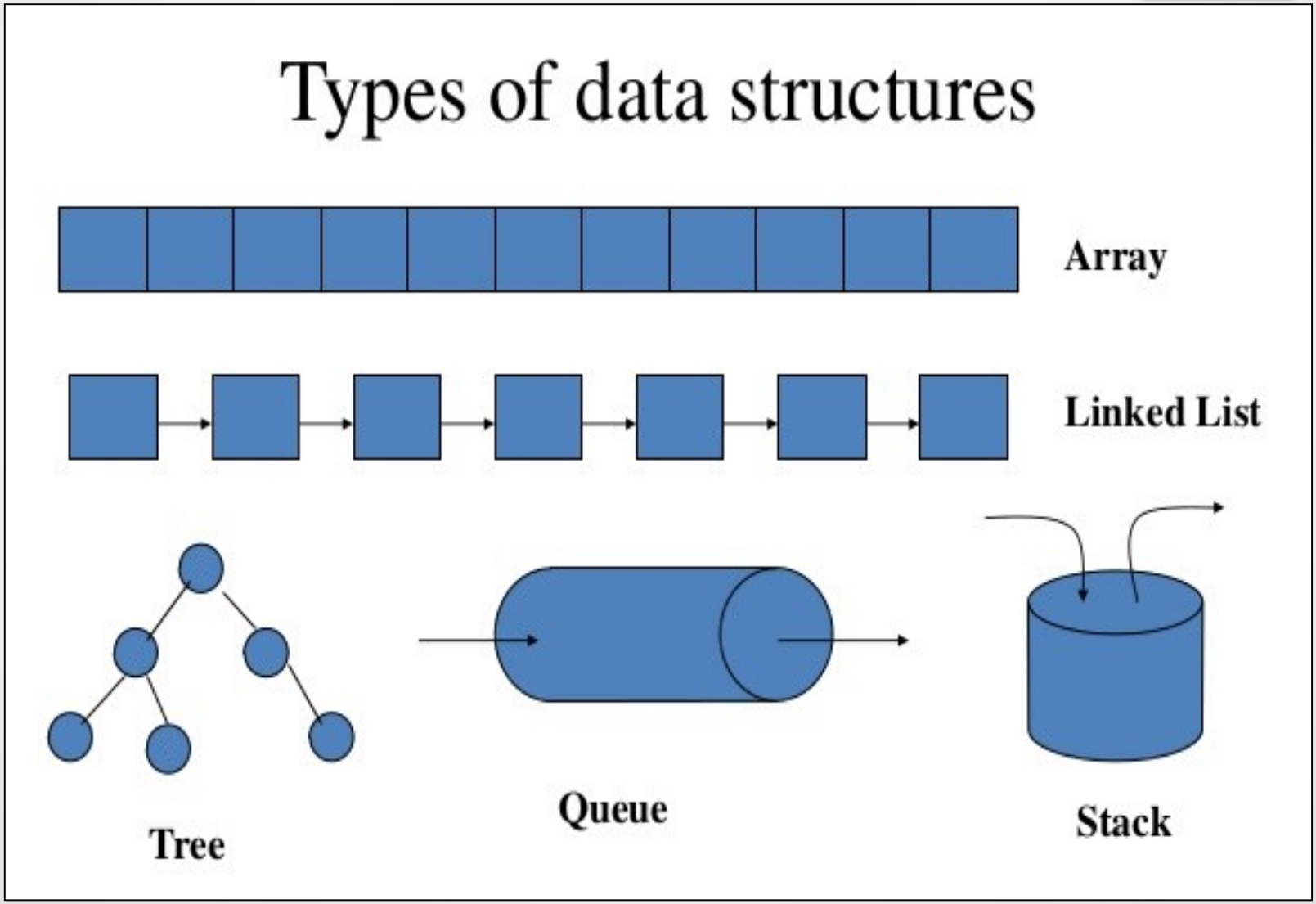


Types of Data Structures: Linear & Non-Linear

- A **linear** data structure traverses the data elements sequentially, in which only one data element can directly be reached. Ex: Arrays, Linked Lists.
- Linked list is an example of linear data storage or structure. Linked list stores data in an organized a linear fashion. They store data in the form of a list.

- **Non-Linear** data structure: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure. Ex: Trees, Graphs.
- Non Linear data structure- Tree data structure is an example of a non linear data structure. A tree has one node called as root node that is the starting point that holds data and links to other nodes.

Types of Data Structures: Linear & Non-Linear



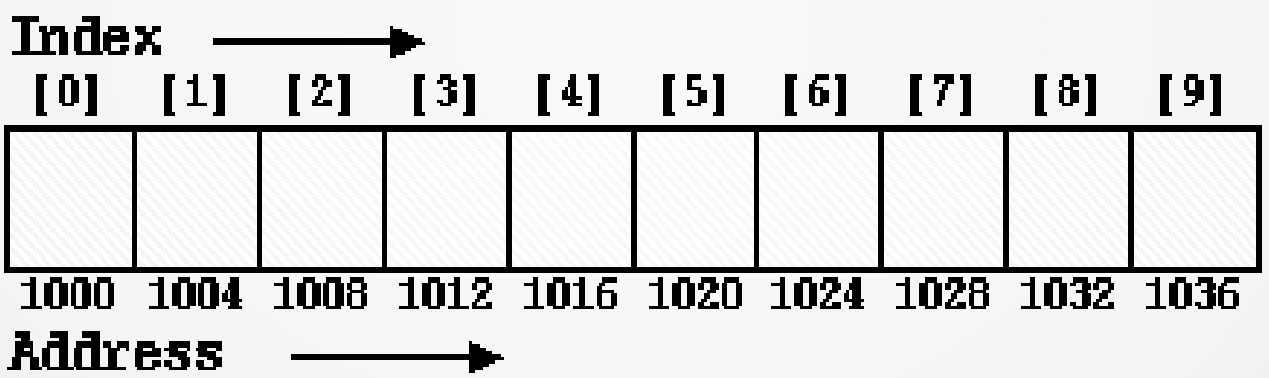
Types of Data Structures: Linear & Non-Linear

- **Linear Data Structure:**
  - An **arrays** is a collection of data elements where each element could be identified using an index.
  - A **linked list** is a sequence of nodes, where each node is made up of a data element and a reference to the next node in the sequence.
  - A **stack** is actually a list where data elements can only be added or removed from the top of the list.
  - A **queue** is also a list, where data elements can be added from one end of the list and removed from the other end of the list.

- **Non-Linear Data Structure:**
  - A **tree** is a data structure that is made up of a set of linked nodes, which can be used to represent a hierarchical relationship among data elements.
  - A **graph** is a data structure that is made up of a finite set of edges and vertices. Edges represent connections or relationships among vertices that stores data elements.

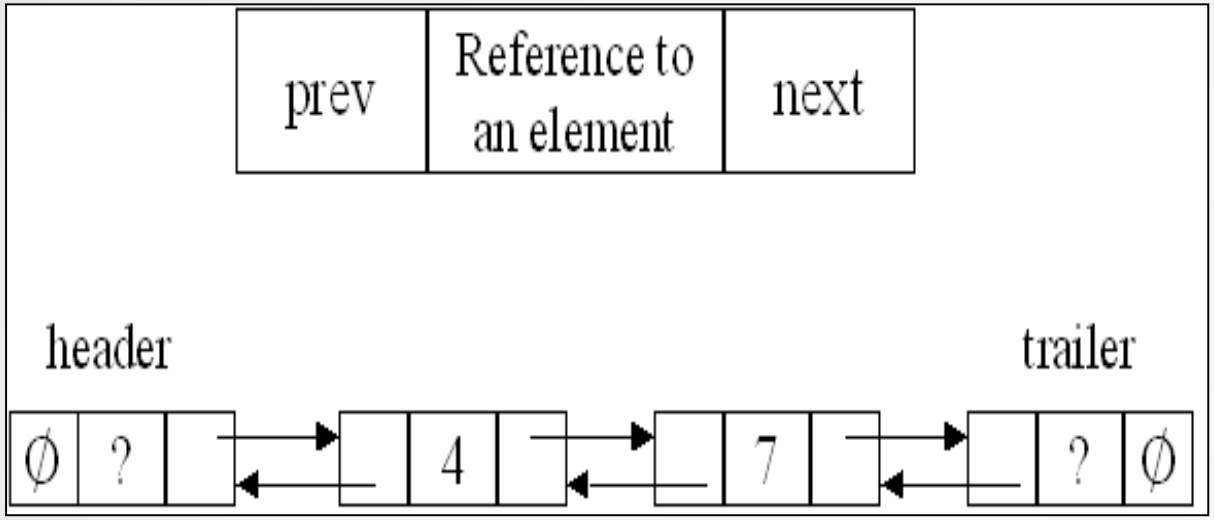
Types of Data Structures: Linear & Non-Linear

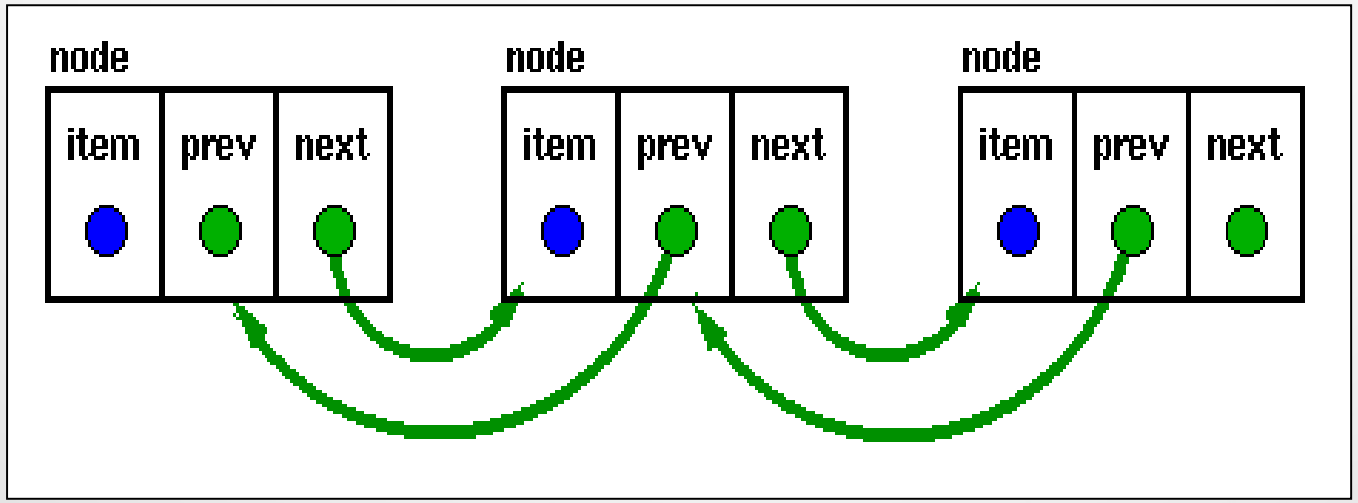
- **In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory. In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them.**



ARRAY STRUCTURE

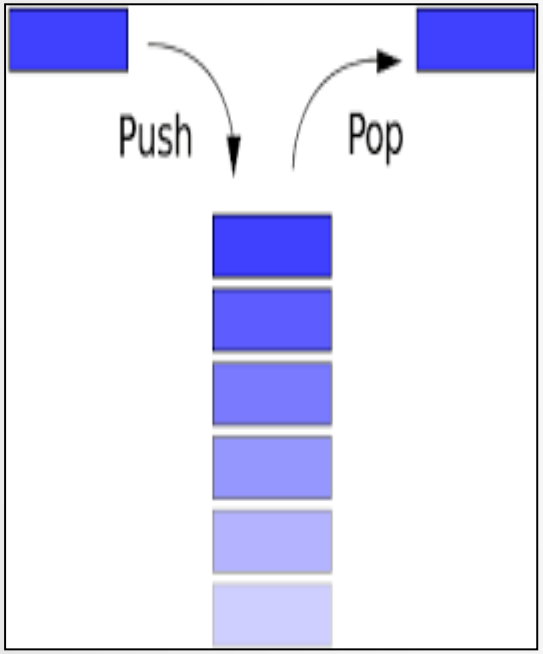
Linear: LINKED LIST

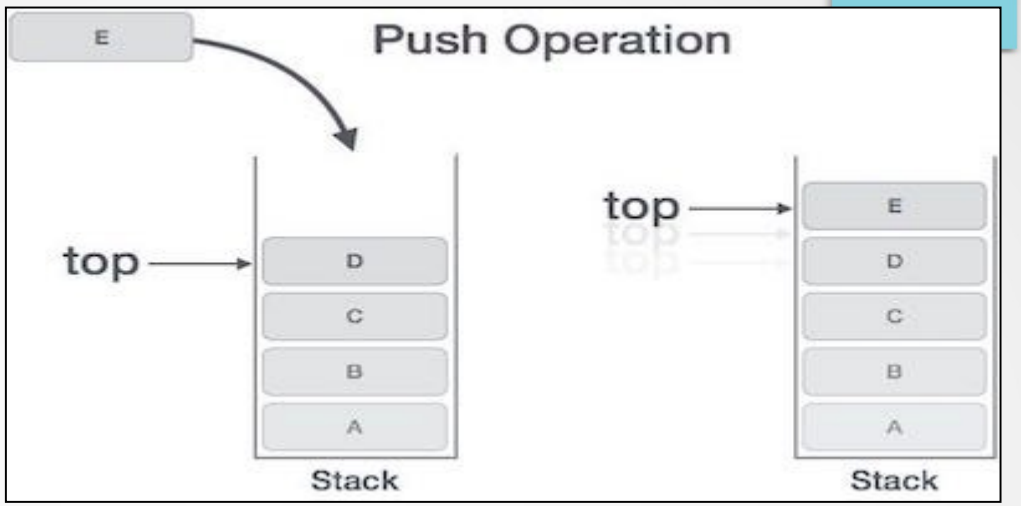


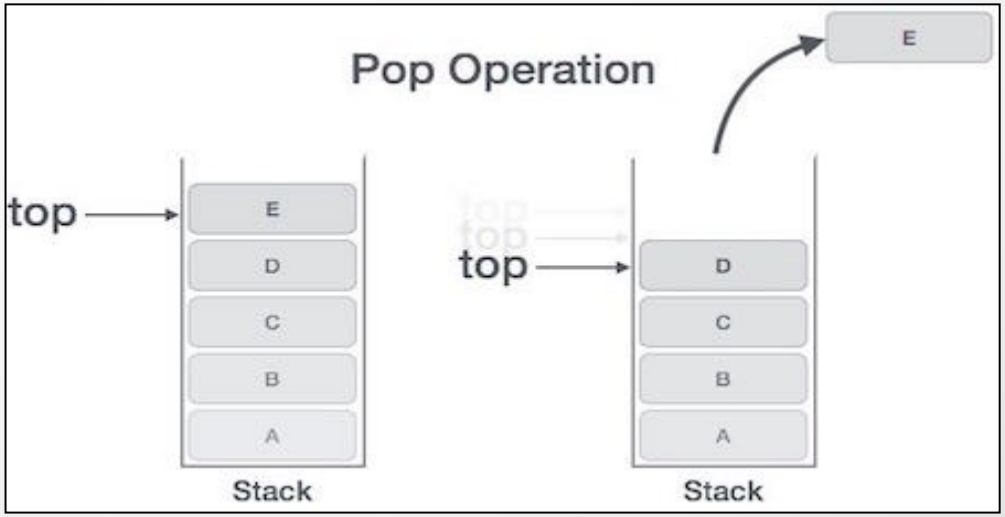


Linked List

Linear: STACK

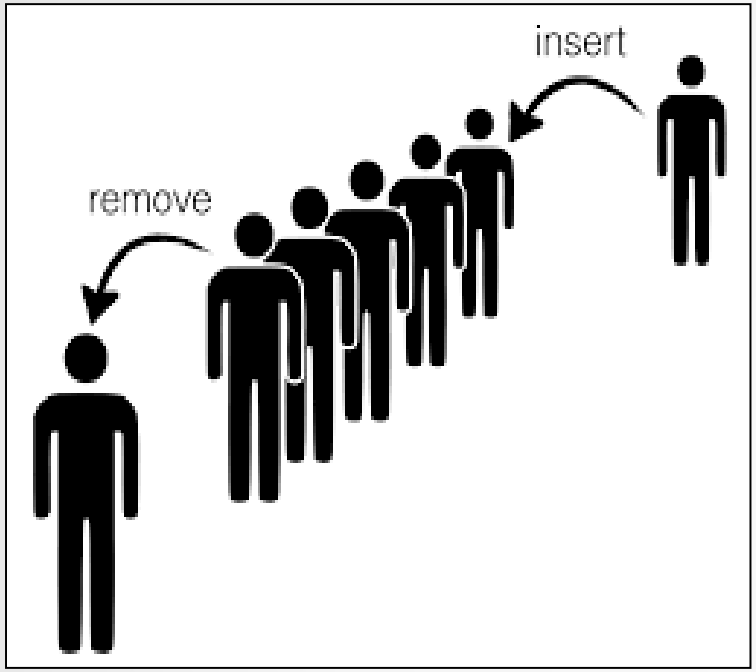


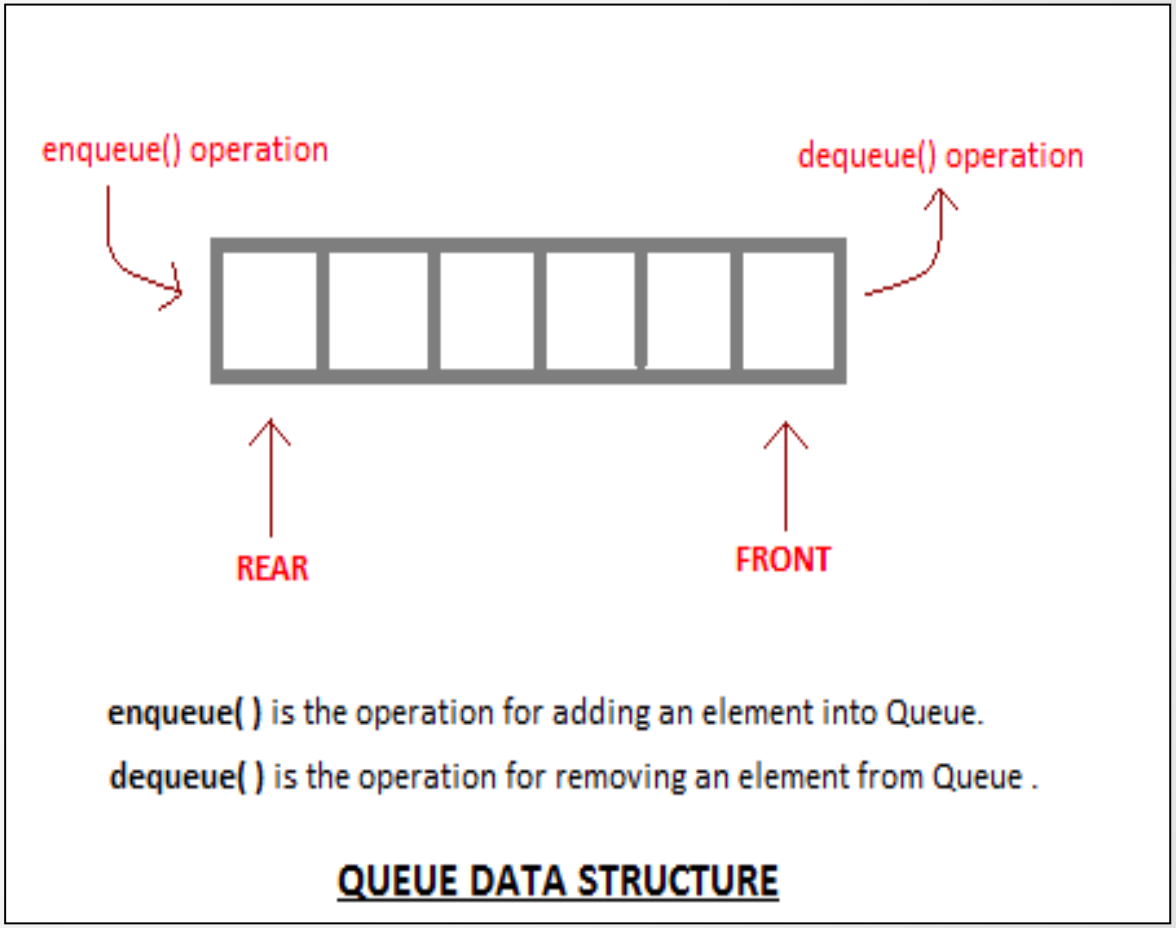




Stack

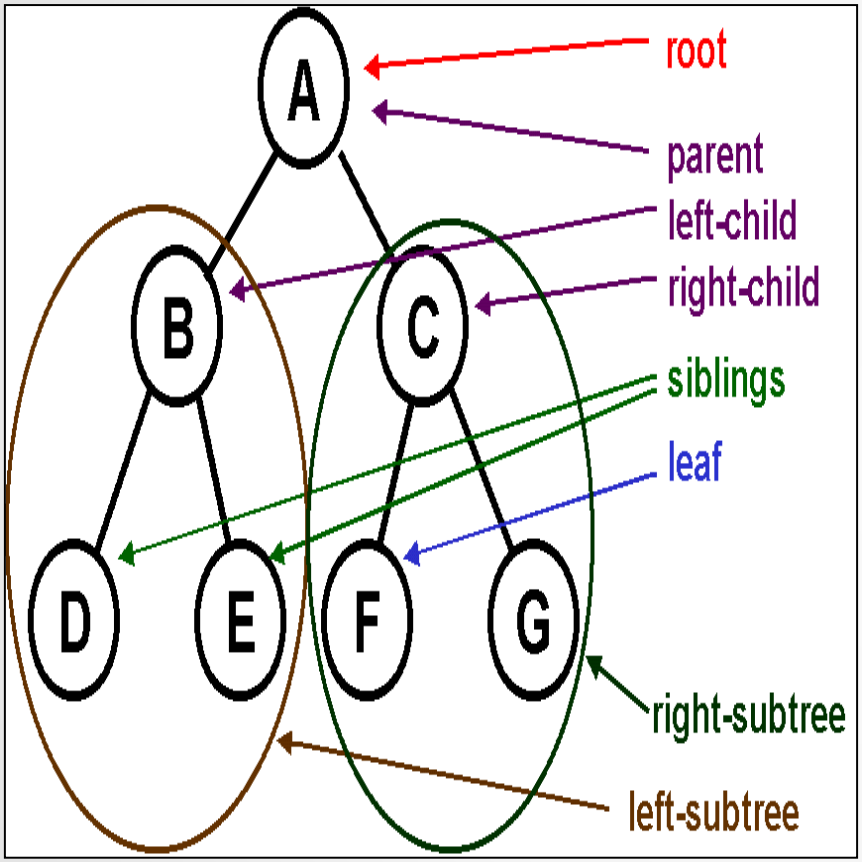
Linear: QUEUE

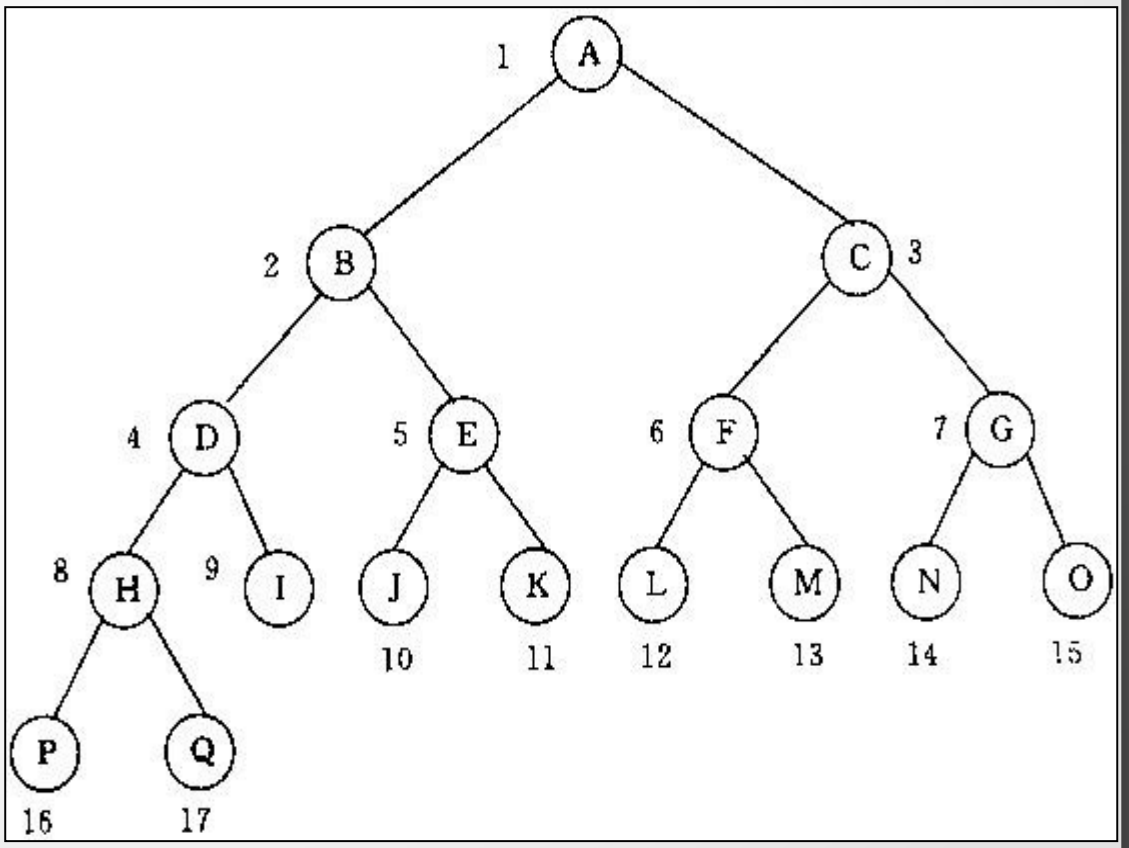




Queue

Non-Linear: TREE



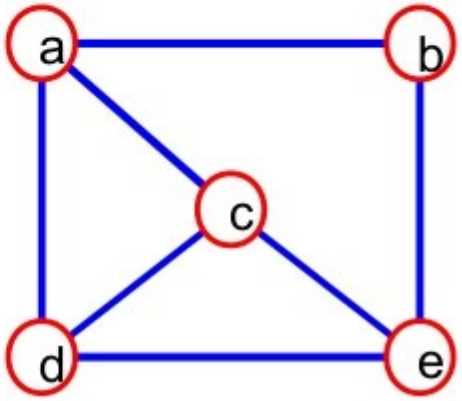


Tree

Non-Linear: GRAPH

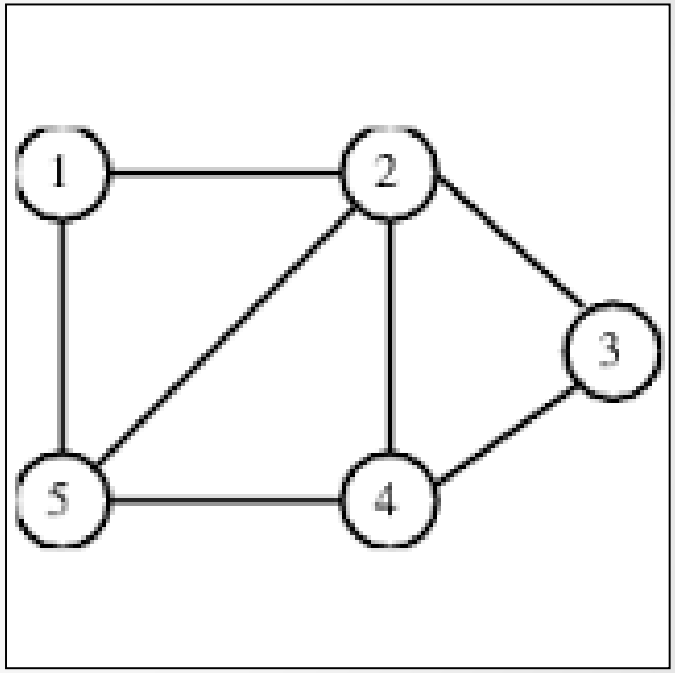
What is a Graph?

- A graph G = (V,E) is composed of:
  V: set of vertices
  E: set of edges connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:



V= {a,b,c,d,e}

E= {(a,b),(a,c),
(a,d),
(b,e),(c,d),(c,e),
(d,e)}



Graph

Arrays

- **An array is a finite ordered collection of homogenous data elements that provides direct access to any of its elements.**
  - **Finite:** number of elements of an array
  - **Ordered Collection:** arrangement of all the elements in an array is specific.
  - **Homogeneous:** All the elements of an array should be of the same data type.
  - **Size of an array:** The maximum number of elements that would be stored in an array is the size of that array. It is also the length of that array.
  - **Base:** The base address of an array is the memory location where the first element of an array is stored.
  - **Datatype of an array:** The data type of an arrya indicates the data type of elements stored in that array.
  - **Index:** A user can access the elements of an array by index/subscript like a[0], a[1]....

Given the base address of an array B[1300…..1900] as 1020 and size of each element is 2 bytes in the memory. Find the address of B[1700].

Solution:

The given values are: B = 1020, LB = 1300, W = 2, I = 1700

Address of A [I] = B + W * (I – LB)

= 1020 + 2 * (1700 – 1300)
= 1020 + 2 * 400
= 1020 + 800
= 1820 [Ans]

Arrays

- **Synatx:** type arrayName [arraySize];

- **Example:** double salary[15000];

- **Initializing an Array:**

```
int age[5]={22,25,30,32,35};
```

OR

```
int newArray[5];
int n = 0;
// Initializing elements of array seperately
for(n=0;n<sizeof(newArray);n++)
{
  newArray[n] = n;
}
```

Two-Dimensional Arrays:

- The simplest form of the multidimensional array is the two-dimensional array.
- It is a list of one-dimensional arrays.
- To declare a two-dimensional integer array of size x,y :
  - type arrayName [x][y];
  - A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array a, which contains three rows and four columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initializing Two-Dimensional Arrays

- Multidimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {
 {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
 {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
 {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

OR

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Sparse Matrix

- A matrix can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represents a mXn matrix. There may be a situation in which a matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as sparse matrix.
- **"Sparse matrix is a matrix which contains very few non-zero elements".**
- **"Sparse matrix is two-dimensional array in which most of the elements are zero".**

Sparse Matrix

- **Types of Sparse Matrix:**
  - Triangular Matrix
    - Lower Matrix
      - Lower Left
      - Lower Right
    - Upper Matrix
      - Upper Left
      - Upper Right
  - Band Matrix
    - Diagonal
    - Tri-diognal