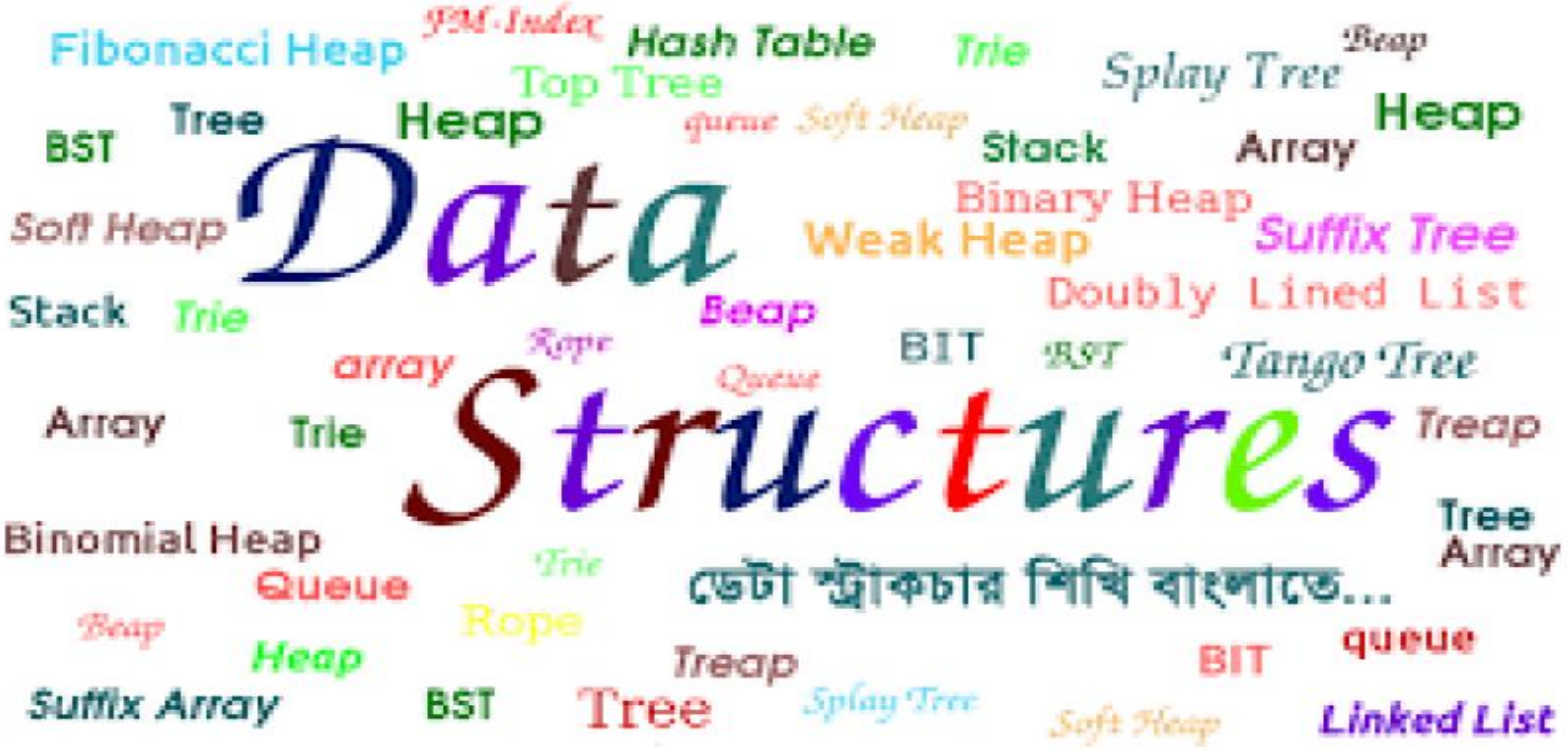
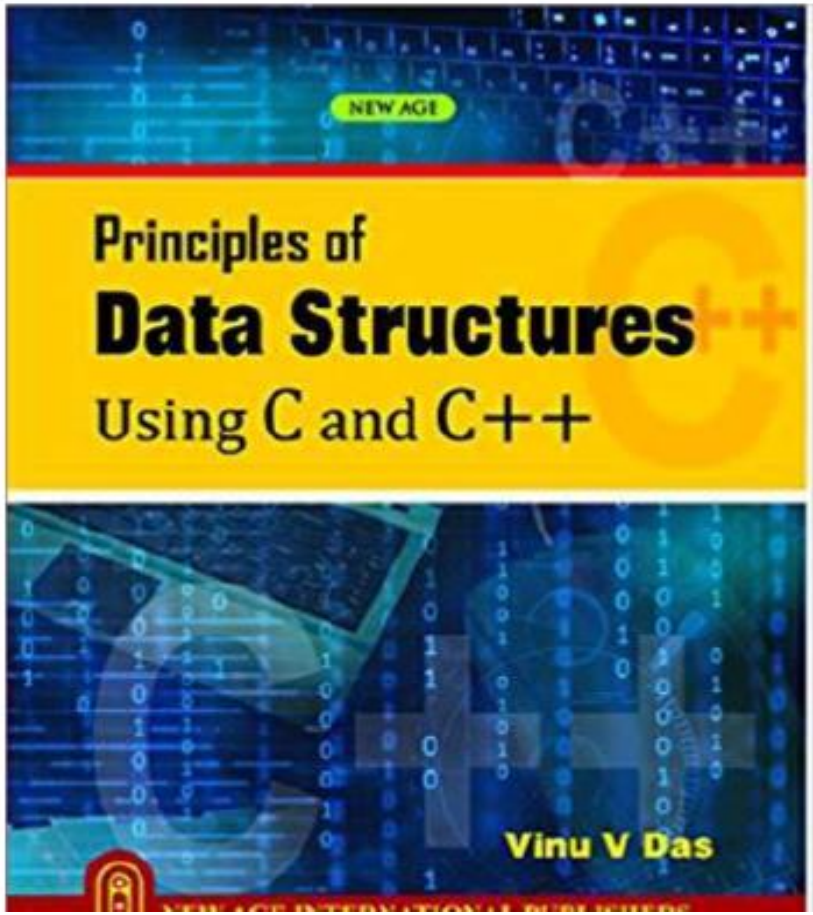

Fibonacci Heap
FM-Index
Hash Table
Trie
Beap
Splay Tree
Top Tree
BST
Tree
Heap
queue
Soft Heap
Heap
Stack
Array
Data
Binary Heap
Soft Heap
Weak Heap
Suffix Tree
Doubly Lined List
Stack
Trie
Beap
Rope
BIT
BST
Tango Tree
array
Queue
Array
Trie
Structures
Treap
Tree
Array
Binomial Heap
Trie
ডেটা স্ট্রাকচার শিখি বাংলাতে...
Queue
Beap
Rope
queue
Heap
Treap
BIT
Suffix Array
BST
Tree
Splay Tree
Soft Heap
Linked List

Book and contents

- Revision on: Arrays/pointers , classes and Structures
- Stack
- Application on Stack
- Recursion - Queue + basic operations functions
- Circular queue + basic operations functions
- De-queue + basic operations functions
- Linear Linked List + basic operations functions
- Circular Linked List + basic operations functions
- Doubly Linked List + basic operations functions
- Tree + basic operations functions



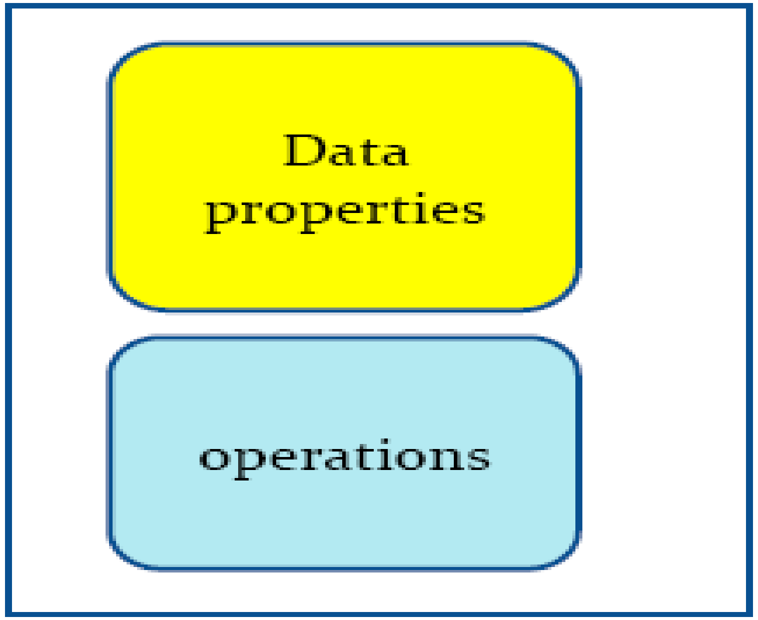
Lecture No. 1:

Introduction to Data Structure

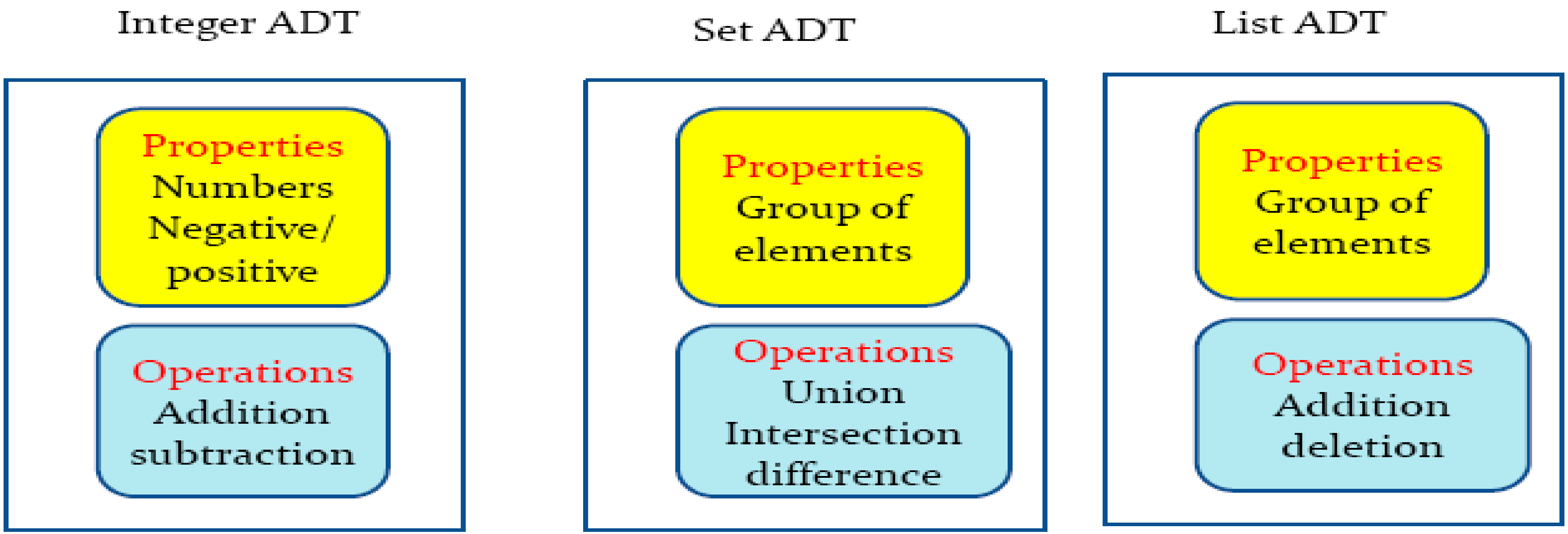
- Abstract Data Type(ADT)
- Data Structure
- Classification of data structure
- Examples of data structure
- Implementation of data structure
- Data structure & Algorithms

Abstract Data Type(ADT)

- **ADT** may be defined as a set of data values and associated operations that are precisely specified independent of any particular implementation.



Abstract Data Type : examples



Integer ADT

Properties
Numbers
Negative/
positive

Operations
Addition
subtraction

Set ADT

Properties
Group of
elements

Operations
Union
Intersection
difference

List ADT

Properties
Group of
elements

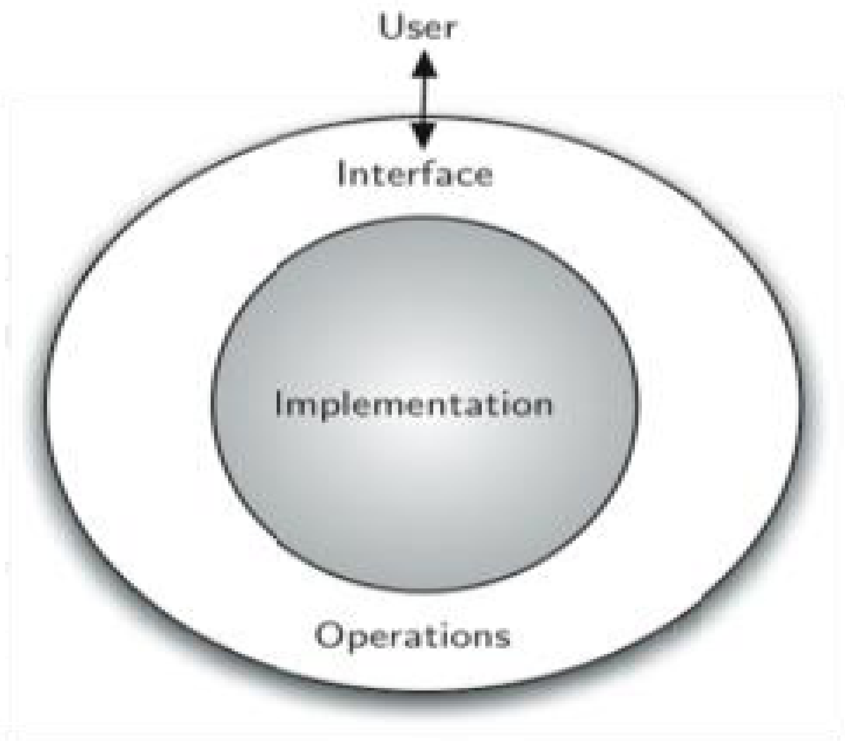
Operations
Addition
deletion

Abstract Data Type

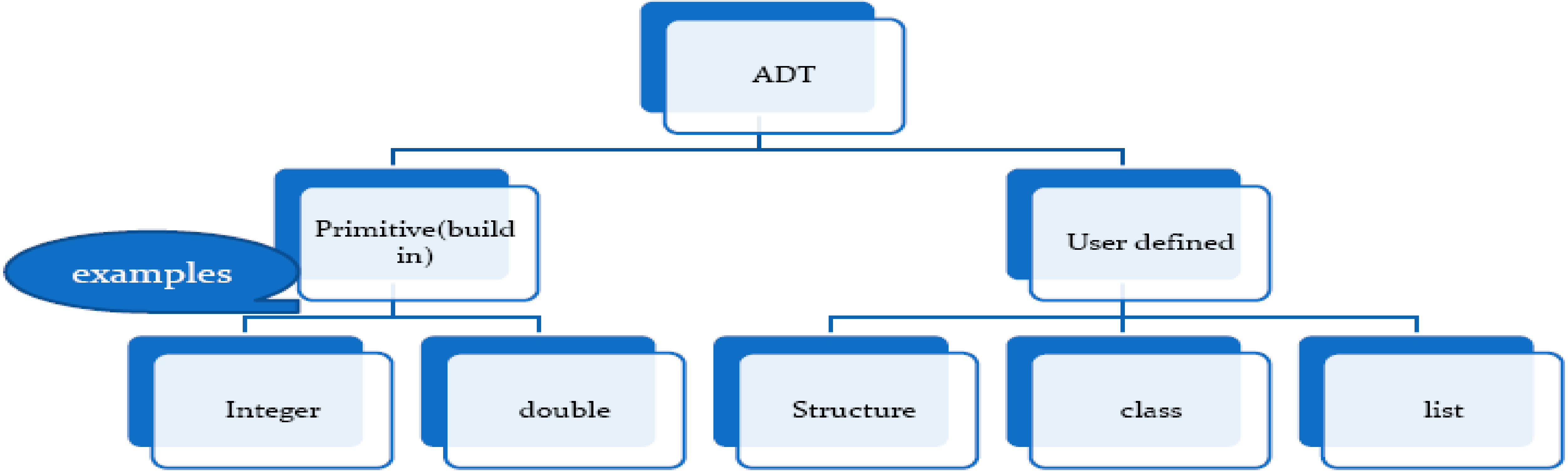
- The definition of ADT only mentions what operations are to be performed but not these operations will be implemented.

- It does not specify how data will be org in memory and what algorithms will be for implementing the operations

- ADT can be considered as a black box hides the inner structure and design of data type



Abstract Data Type



Data Structure

- Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way
- It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.
- An example of several common data structures are:
  - arrays
  - Stacks
  - Queues
  - linked lists
  - Trees.
  - Graphs.
  - hash tables.

ADT & Data structure

- ADT is a logical description and data structure is concrete
- ADT is implementation independent and data structure is implementation dependent

Example :

- ADT only describes what a data type List consists (data) and what are the operations it can perform, but it has no information about how the List is actually implemented. But data structure tell us how the List implemented i.e., using array or linked list

Why data structure?

Why data structure?

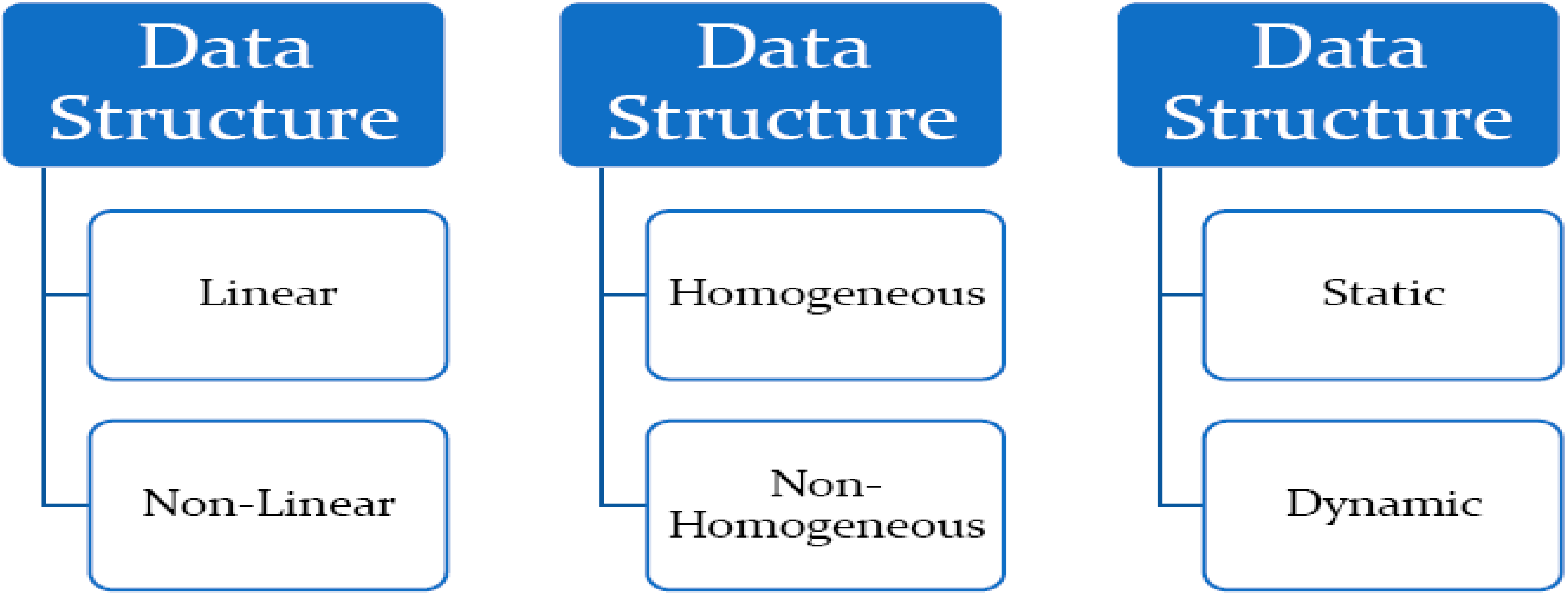
- Data structures are important for the following reasons:

1. Data structures are used in almost every program or software system.

2. Specific data structures are essential ingredients of many efficient algorithms, and make possible the management of huge amounts of data, such as large integrated collection of databases.

3. Some programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design.

Classification of data structure



Data Structure

- Linear
- Non-Linear

Data Structure

- Homogeneous
- Non-Homogeneous

Data Structure

- Static
- Dynamic

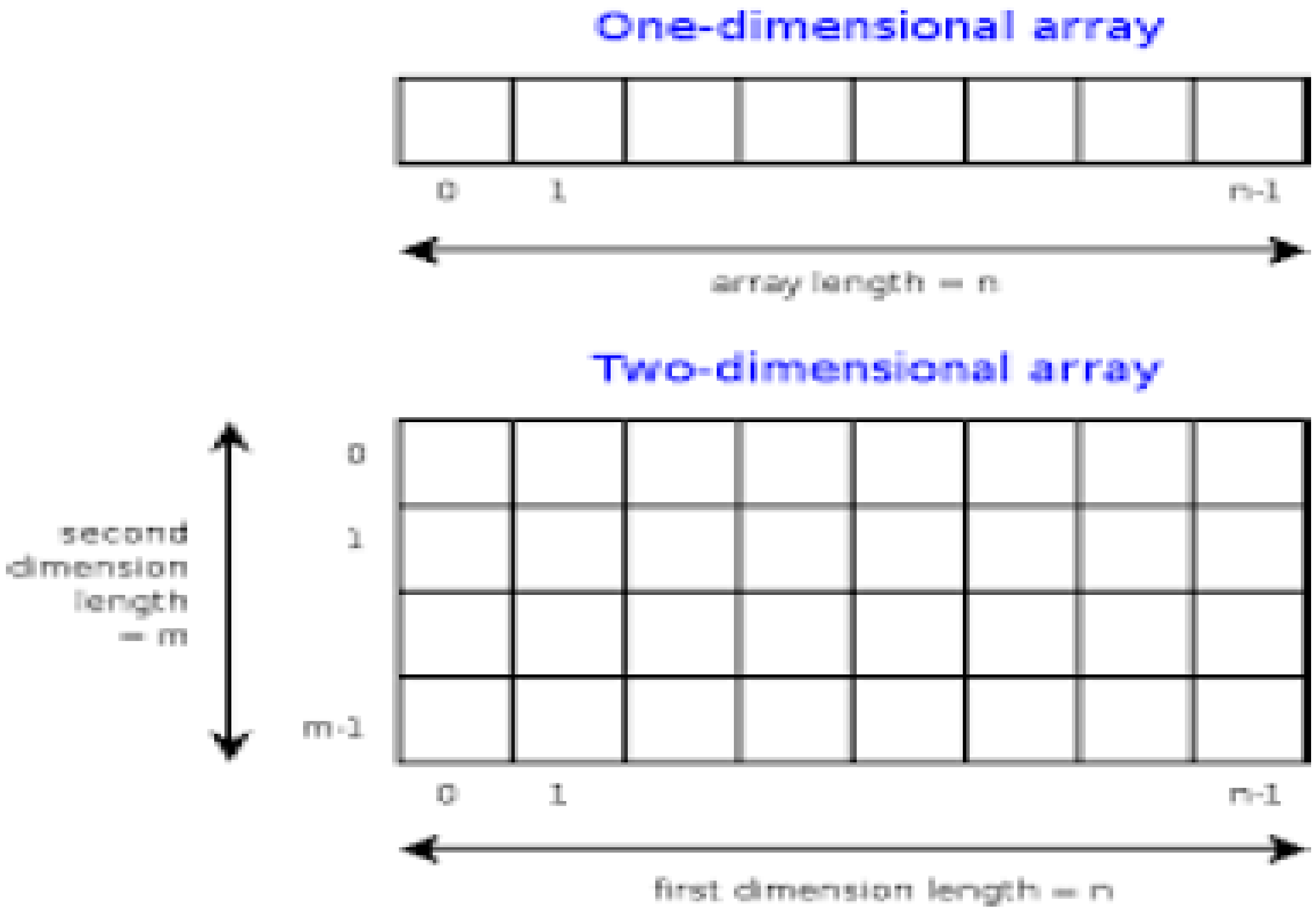
Classification of data structure

Description	Characteristic
Linear	In Linear data structures, the data items are arranged in a linear sequence. Example: **Array, stack, queue**
Non-Linear	In Non-Linear data structures, the data items are not in sequence. Example: **Tree**, **Graph**
Homogeneous	In homogeneous data structures, all the elements are of same type. Example: **Array**
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Structures**
Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array**
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: **Linked List created using pointers**

Data structure : Array

An array is an aggregate data structure that is designed to store a group of objects of the same types. Arrays can hold primitives as well as references.

The array is the most efficient data structure for storing and accessing a sequence of objects.

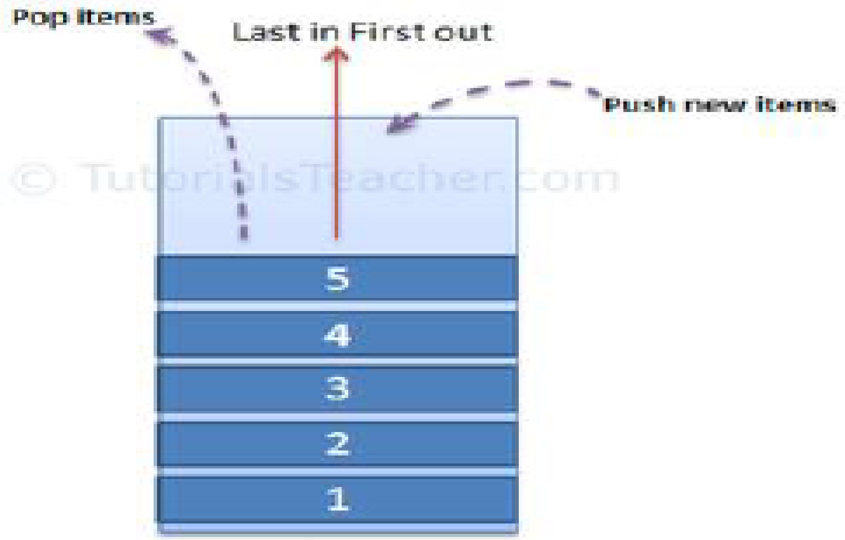


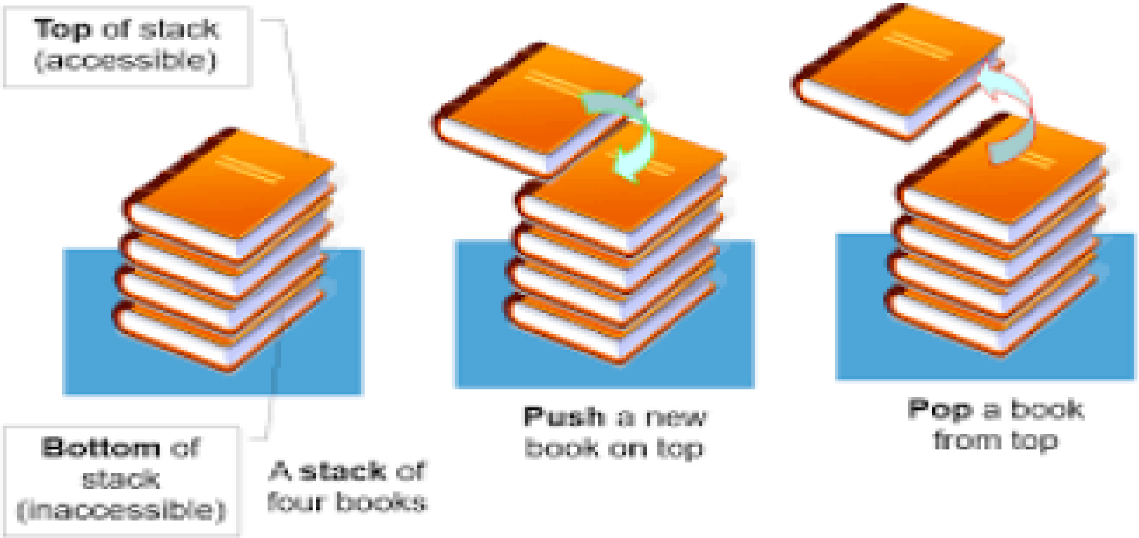
Data structure : queue

- **A Queue** is a linear structure which follows a particular order in which the operations are performed. The order is **First In First Out (FIFO).**

LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT

Data structure : Stack

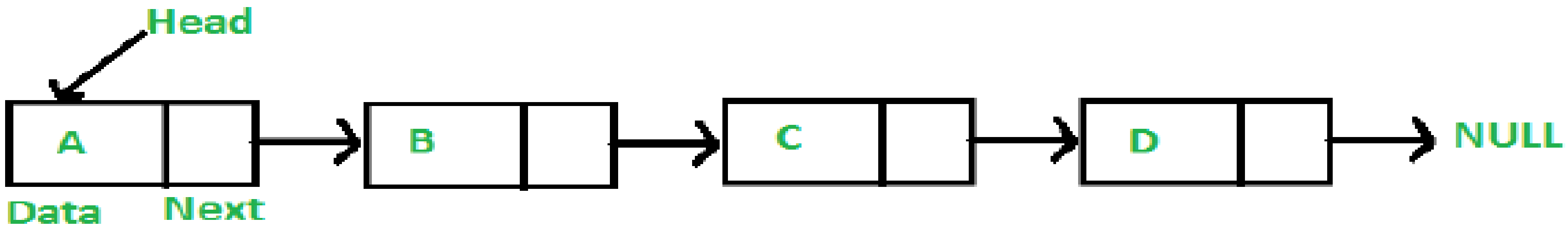




<span style="color:red">Stacks</span> are data structures that follow the **Last In First Out (LIFO)** principle. The last item to be inserted into a stack is the first one to be deleted from it.

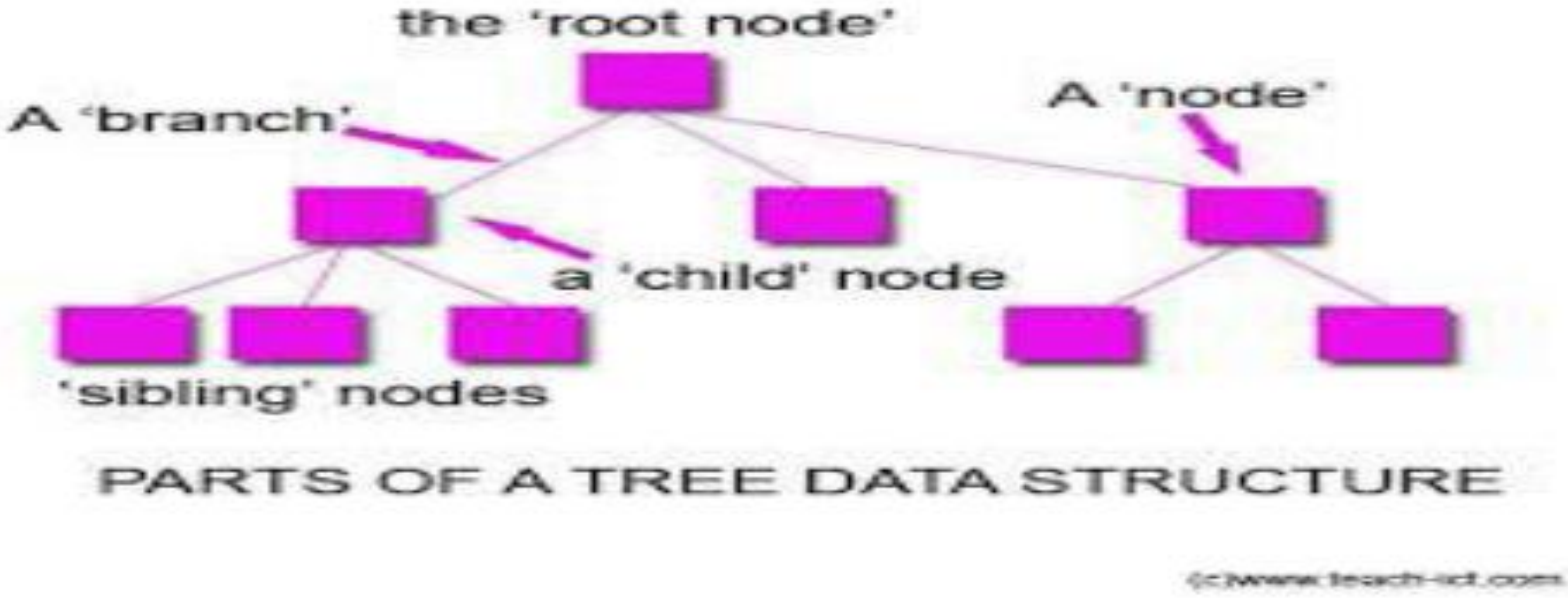
Data structure :Linked List

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers
- a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.



Data Structure: Tree

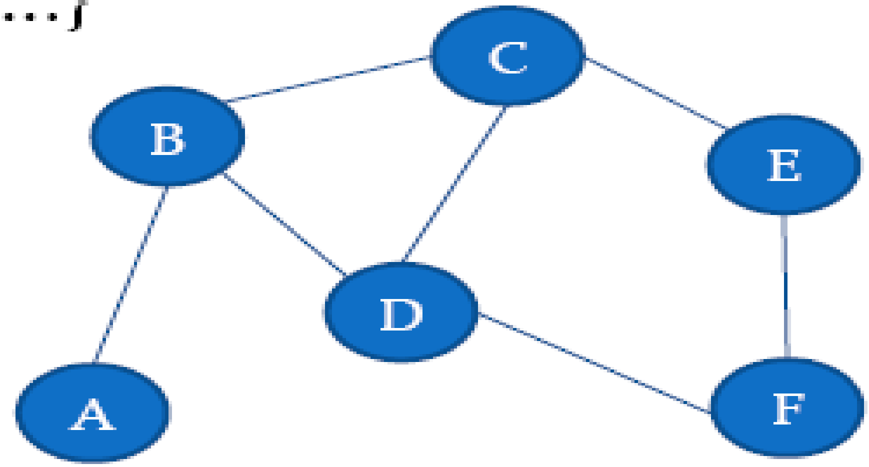
- A tree is a *nonlinear* data structure,
- A tree is a collection of nodes connected by directed (or undirected) edges
- A tree is a structure consisting of one node called the **root** and zero or one or more subtrees.



PARTS OF A TREE DATA STRUCTURE

Data Structure : Graphs

- **Graphs** representations have found application in almost all subjects like geography, engineering and solving games and puzzles.
- A graph G consist of
- 1. Set of vertices V (called nodes), (V = {v1, v2, v3, v4......}) and
- 2. Set of edges E (i.e., E {e1, e2, e3......}



Data structure which one?

How much data you need to store?

How often do you access data?

What is the purpose of the data? Do you need to access, insert, delete, or sort the data?

THANK
YOU

Any
Question?