CSS3122 DATA STRUCTURE AND ALGORITHMS

CH1 INTRODUCTION TO DATA STRUCTURE

KHAIRUNNISA IBR

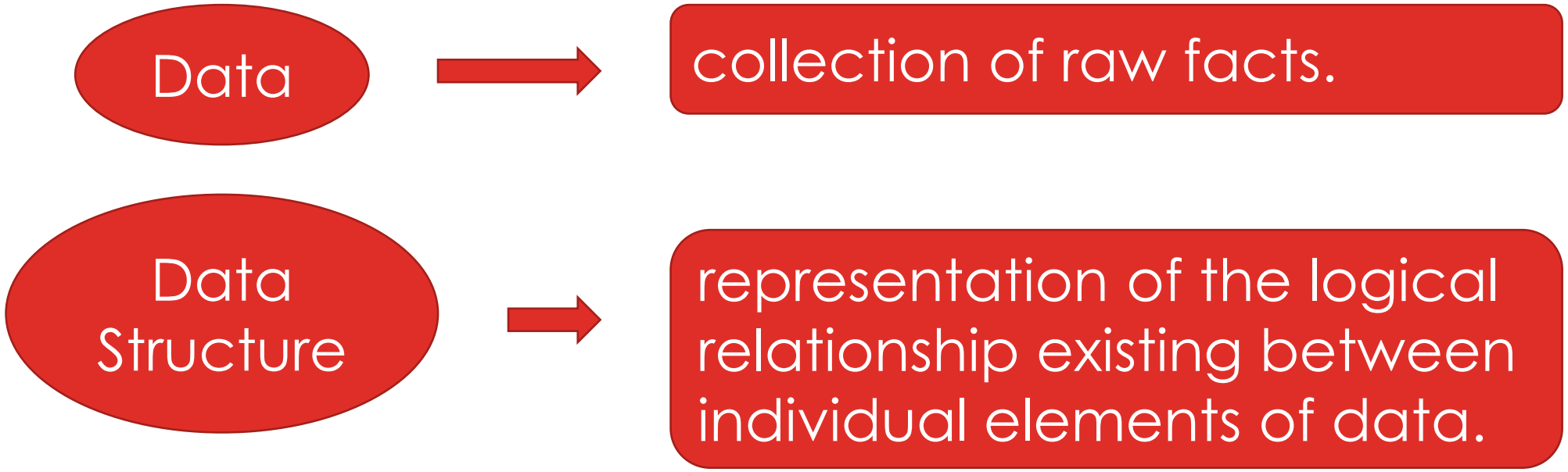
CHAPTER OUTLINE

In this chapter, you will learn about:

1. Data Structure Concept
2. Basic types of Data Structures
3. Primitive Data Structures
4. Non-Primitive/Abstract Data Structure (ADT)
5. Need of Data Structure
6. Advantages of Using Data Structure

DATA STRUCTURE CONCEPT

DEFINITION



- **Data structure** is a specialized format for organizing and storing data in memory that considers not only the elements stored but also their relationship to each other.

DATA STRUCTURE CONCEPT (CONT)

- Data structure affects the design of both structural & functional aspects of a program.

Program = Algorithm + Data Structure

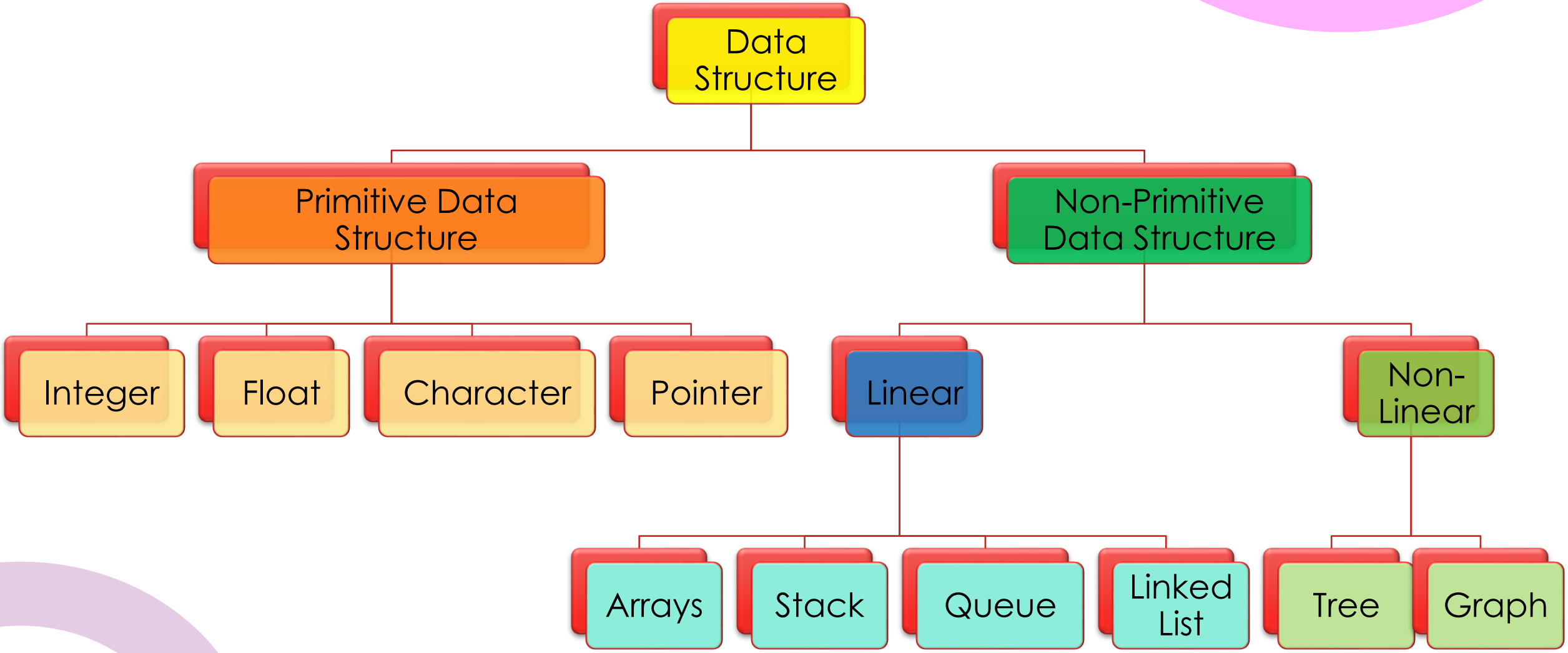
- You knew that an algorithm is a step-by-step procedure to solve a particular function.

BASIC TYPES OF DATA STRUCTURE (CONT)

- Data structure are normally divided into 2 broad categories:

1. **Primitive Data Structure**
2. **Non-primitive Data Structure**

BASIC TYPES OF DATA STRUCTURE (CONT)



PRIMITIVE DATA STRUCTURE

- Data structures that are directly operated upon the machine-level instructions.
- Can hold a single value in a specific location.
- Consists of fundamental data types such as
  - integer,
  - float,
  - character, and
  - boolean.

PRIMITIVE DATA STRUCTURE (CONT)

Some e.g. of primitive data structures:

- Integer - contains numeric values. whole numbers that can be either negative or positive. When the range of integer data type is not large enough then in that case, we can use long.
- Float - a data type that can hold decimal values. When the precision of decimal value increases then the Double data type is used.
- Character - a data type that can hold a single character value both uppercase and lowercase such as 'A' or 'a'.
- Boolean - a data type that can hold either a True or a False value. It is mainly used for checking the condition.

PRIMITIVE DATA STRUCTURE (CONT)

- Common operations done on primitive data structures:

Create	• Allocating a memory location for the structure.
Selection	• Selecting the memory location to be used in further action.
Update	• Change value stored in the structure.
Destroy	• Remove the value stored in the structure and delete the reservation of the memory location for the structure.

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

- Also called as abstract data structure (ADT).
- A derivation from primitive data structure.
- Can hold multiple values either in a contiguous location or random locations.

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

- Divided into 2 types:

1. Linear Data Structure

- A sequential type of data structure.
- E.g. array, stack, queue, linked list.

2. Non-Linear Data Structure

- A random type of data structure.
- E.g. tree, graph.

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

- Operation that can be done on a data structure.

Traversing	•Accessing each record exactly once so that certain item in the record is processed.
Searching	• Finding the location of the record with a given key value.
Insertion	• Add a new record to the structure.
Deletion	• Removing a record from the structure.
Selection	• Selecting the memory location to be used in further action.
Sorting	• Arranging values in the structure in certain order.
Merging	• Combining values into a structure.

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

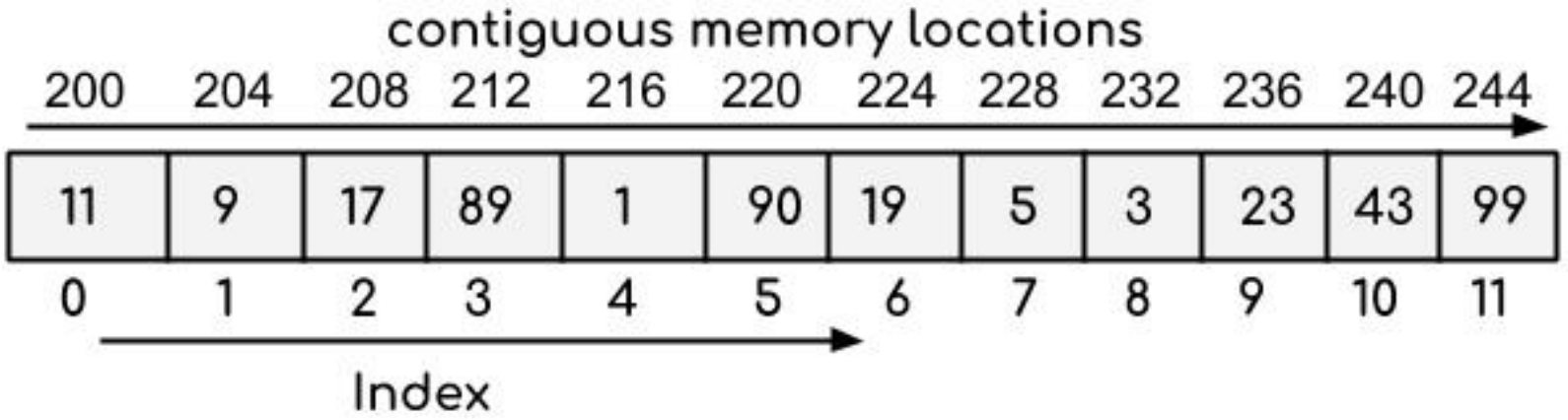
LINEAR ADT

1. Array
   - An array is a collection of homogenous type of data elements.
   - An array is consisting of a collection of elements.
   - Operation performed on array:

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

1. Array (cont…)
   - Representation of array in memory:



NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

2. Stack
   - A stack is **a list of elements in which an element may be inserted or deleted at one end** which is known as TOP of the stack.
   - Operation performed on stack:

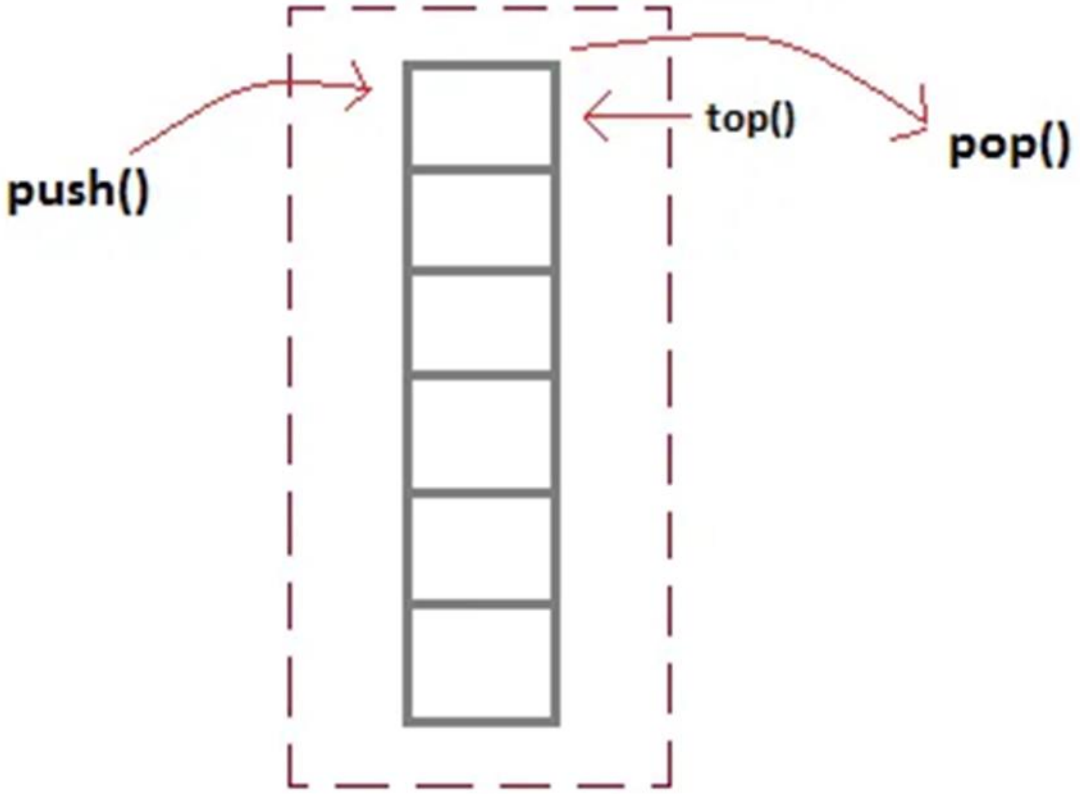
Push: add an element in stack

Pop: remove an element in stack

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

2. Stack (cont…)
   - Representation of stack in memory:



NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

3. Queue
   - A queue is a linear list of element in which insertion can be done at one end which is known as front and deletion can be done which is known as rear.
   - Operation performed on queue:

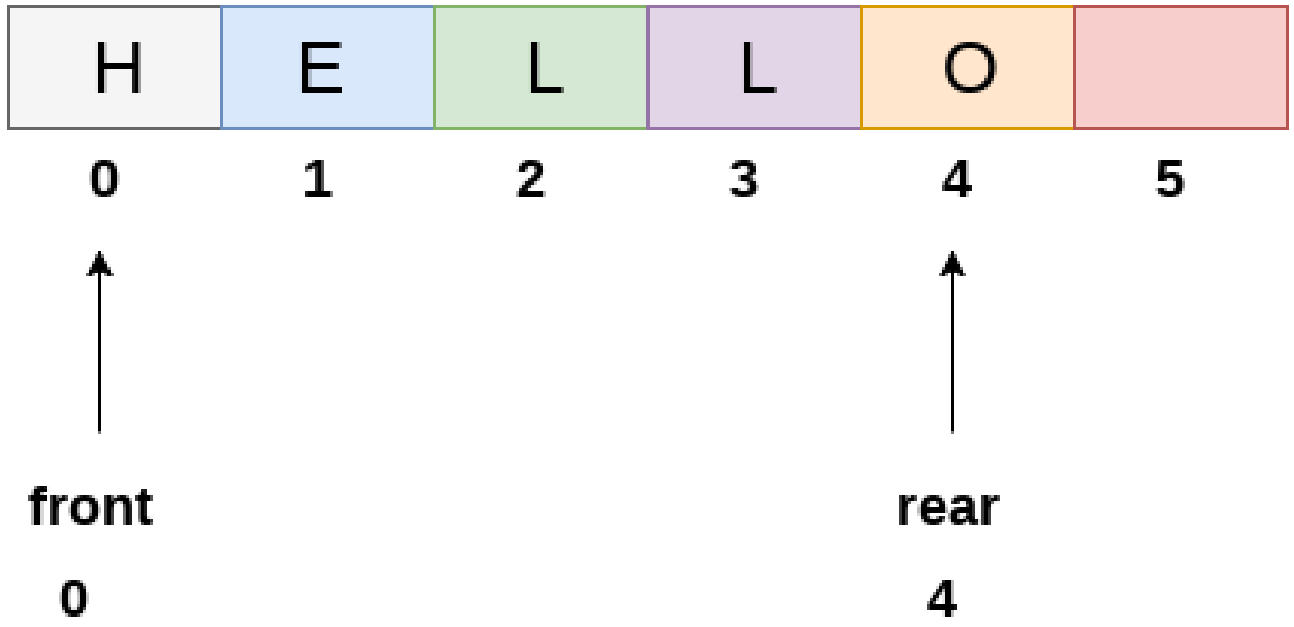
Insertion: add a new element in queue

Deletion: Removing an element in queue

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

3. Queue (cont…)
   - Representation of queue in memory:



NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

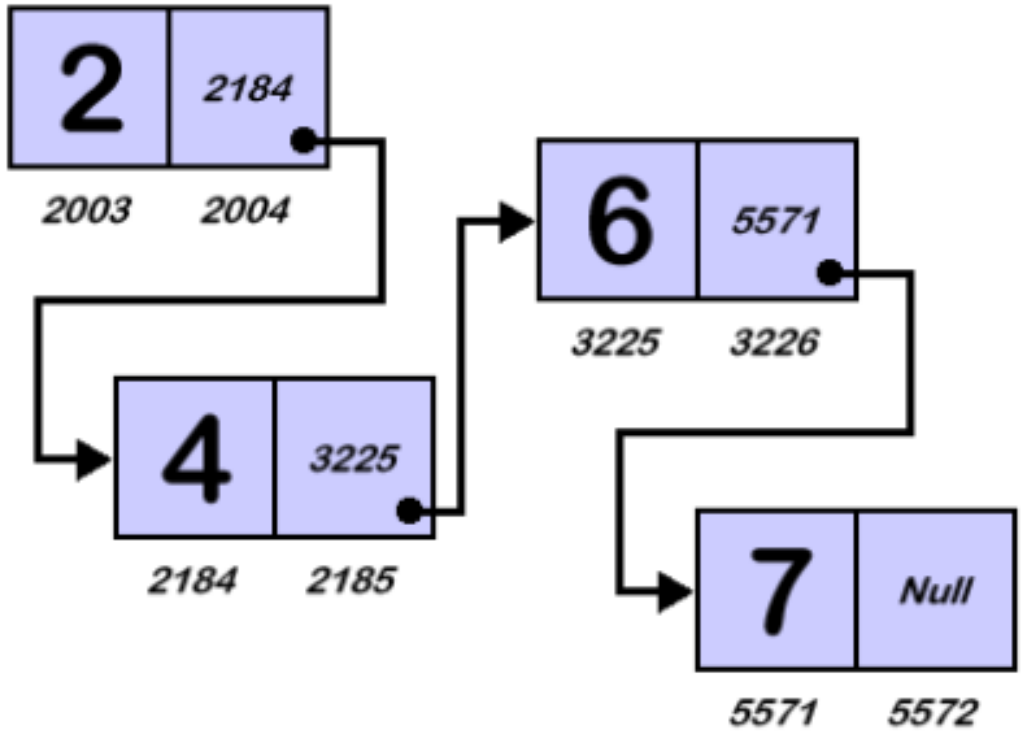
4. Linked List
   - A Linked list is a linear collection of data elements.
   - It has two part one is info and other is link part.
   - Info part gives information and link part is address of next node.
   - Operation performed on linked list:

Traversing | Search | Insertion | Deletion

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

LINEAR ADT (cont)

4. Linked List (cont…)
   - Linked list representation



NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

NON-LINEAR ADT

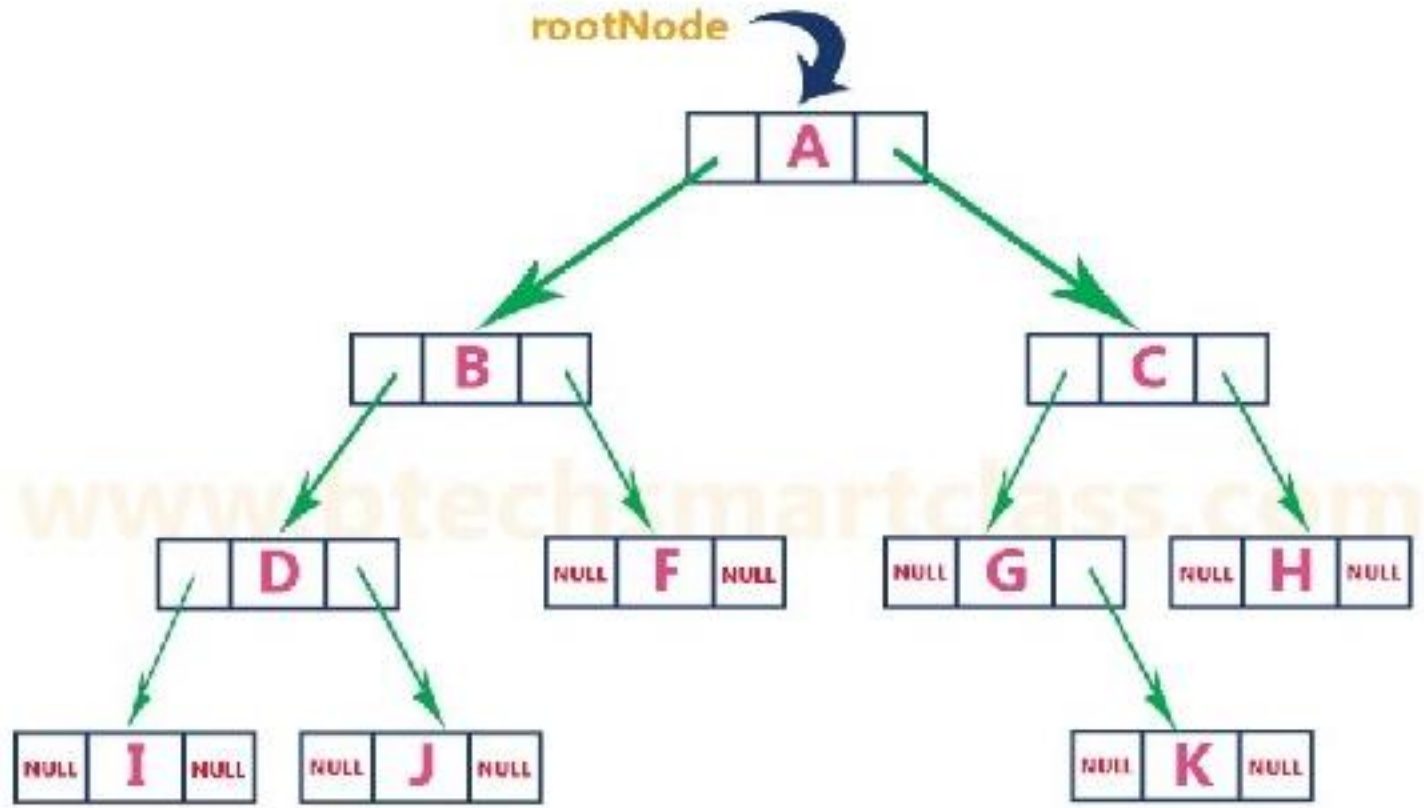
1. Tree
   - In computer science, a tree is a widely-used data structure that emulates a hierarchical tree structure with a set of linked nodes.
   - Operation on tree:

Insertion | Deletion | Searching

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

NON-LINEAR ADT

1. Tree (cont…)
   - Tree representation:



NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

NON-LINEAR ADT

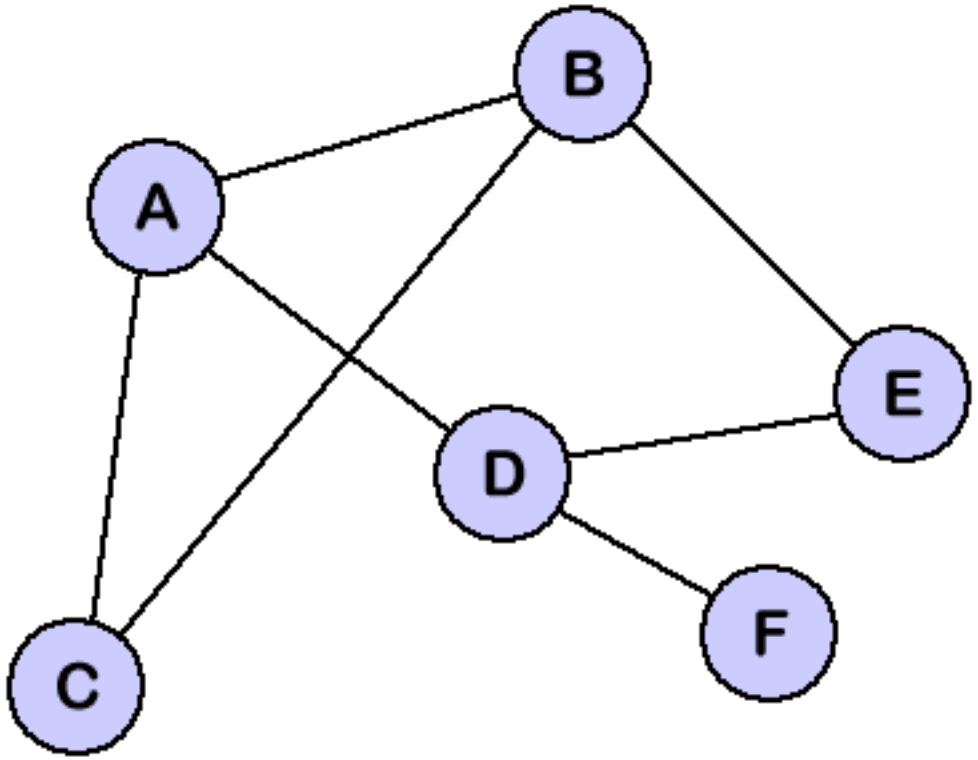
2. Graph
   - A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.).
   - Operation on graph :

Insertion | Deletion | Searching

NON-PRIMITIVE/ABSTRACT DATA STRUCTURE (ADT)

NON-LINEAR ADT

2. Graph (cont…)
   - Graph representation:



PRIMITIVE VS. NON-PRIMITIVE DATA STRUCTURE

Primitive Data Structure	Non-Primitive Data Structure
stores the data of only one type.	can store the data of more than one type.
examples are integer, character, float.	examples are Array, Linked list, stack.
will contain some value, i.e., it cannot be NULL.	can consist of a NULL value.
size depends on the type of the data structure.	size is not fixed.
can be used to call the methods.	cannot be used to call the methods.

NEED OF DATA STRUCTURE

- As applications are getting more complex and the amount of data is increasing, these problems may arise:

Processor speed	• High speed processing is required to process large data, but data keep increasing.
Data search	• Large data need to be arranged properly so that they can be searched efficiently with minimal time.
Multiple requests	• Large server can fail if thousands of users are searching for the same data simultaneously.

ADVANTAGES OF USING DATA STRUCTURE

- The following are few advantages of data structure:

Efficiency	• Searching process can be very efficient depending on the way data is organized and the searching algorithms used (e.g. binary search tree algorithm).
Reusability	• Once data structure is implemented, it can be compiled into libraries to be used at any other place by different clients.
Abstraction	• Client program uses data structure through interface only, without getting into the implementation details.

THAT'S ALL FOR NOW…

- See you next week!