Data Structures and Algorithms

- Difference between “Information” and “Data”

  - The quantities, characters or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical or mechanical recording media.

    Example: c = a + b ⇒ Here, a and b are data.

  - If data is arranged in a systematic way, then it gets a structure and becomes meaningful. This meaningful or processed data is called **Information**.

    Example: DATA ⇒ TEERPSAJ SI EMAN YM (just a collection of characters)
    INFORMATION ⇒ MY NAME IS JASPREET

- To provide an appropriate way to structure the data, we need to know about DATA STRUCTURES.

- A **Data Structure** is the systematic way to organize data so that it can be used efficiently.
  Example: Arrays, Linked Lists

- **Real life examples of Data Structures:**
  - Undo/Redo feature ⇒ Stack is used for this feature
  - Store an image as a bitmap ⇒ Array is used for this purpose
  - Storing friendship information on a Social Networking Site ⇒ Graph is used for this purpose

- Two important things about data types:
  - Defines a certain **domain** of values
  - Defines **Operations** allowed on those values

- **Abstract Data Types** are like user-defined data types which define operations on values using functions without specifying what is there inside the function and how the operations are performed.

  Example: Stacks

- Data structures are used to implement ADTs.
- ADT tells us **what** is to be done and data structures tell us **how** to do it.
- ADT is the **blueprint** while Data Structure is the implementation.

- Advantages of Data Structures:
  - Efficiency
  - Reusability
  - Abstraction

- A data structure is linear when all the elements are arranged in a linear (sequential) order.
  Example: Arrays, Linked Lists, Stacks, Queues

- A data structure is non-linear when all the elements are not arranged in a linear (sequential) order. There is no linear arrangement of the elements.

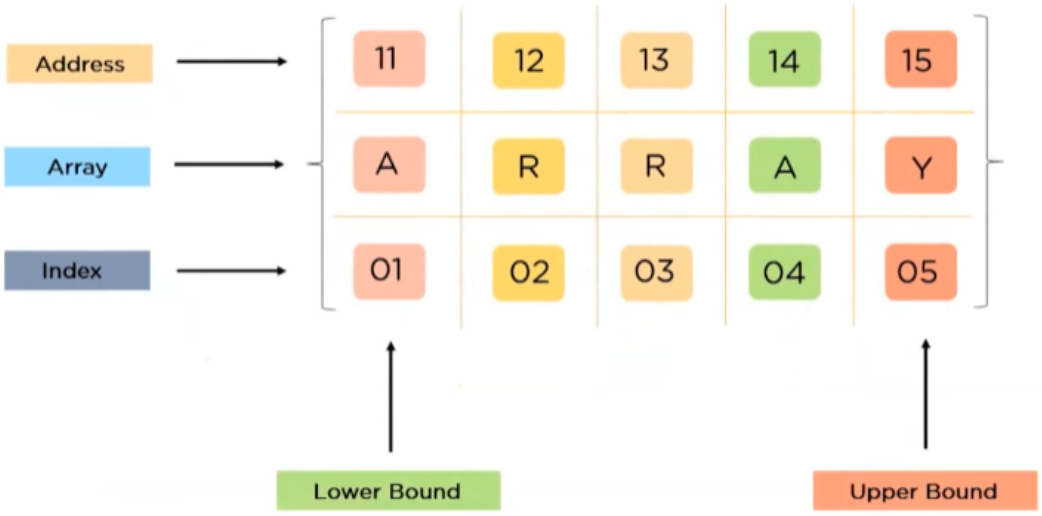
- Static Data Structures:
  - In these types of data structures, the memory is allocated at compile time. Therefore, the maximum size is fixed.

    Example: Arrays
    Advantage: Fast access
    Disadvantage: Slower insertion and deletion

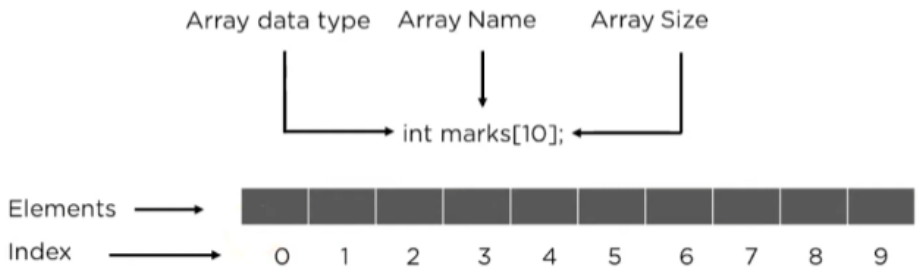
- Dynamic Data Structures:
  - In these types of data structures, the memory is allocated at run time. Therefore, the maximum size is flexible.

    Example: Linked Lists
    Advantage: Faster insertion and deletion
    Disadvantage: Slower access

- An **Array** is a linear data structure that collects elements of the same data type and stores them in contiguous and adjacent memory locations.



- Types of Arrays:
  - One-dimensional arrays : require only one subscript to specify elements in an array.



  - Multi-dimensional arrays (2D and 3D Arrays): require more than one subscript to specify elements in an array.

- **Stack:**
  - an abstract data type
  - follows LIFO (Last-In-First-Out)
  - two operations : Push and Pop

- Efficiently implemented using arrays
- An element gets inserted at the top of the stack when “push”(ed) and the top element gets removed and returned when “pop”(ed).

➙ **Queue:**
- an abstract data type
- follows FIFO (First-In-First-Out)
- Two operations : Push and Pop
- Efficiently implemented using Linked Lists
- An element gets inserted at the end of the queue when “push”(ed) and the first element gets removed and returned when “pop”(ed).

➙ **Deque:**
- An element can be inserted at the back of the queue as well as front of the queue. Similarly, an element can be removed from the back as well as the front of the queue. (It thus can be considered a stack + queue Data Structure)

➙ **Linked Lists:**
- Unlike arrays, Linked Lists are not contiguous in the memory.
- Single Linked List ⇒ Navigation is forward only
- Doubly Linked List ⇒ Forward and backwards navigation is possible
- Circular Linked List ⇒ Last element is linked to the first element.

➙ **Single Linked List**
- Each node in the list consists of 2 things : A pointer and a value
- Value part of the node stores the value associated with the node.
- Pointer stores the memory address of the next node in the list.
- The first node of the list is termed as “Head” or “Root” of the list. A pointer is used to refer to the very first node of the linked list.
- The last node of the list is said to be “Tail” of the list.

➙ **Doubly Linked List**
➙ **Circular Linked List**