Agenda

- Definition of Data structure
- Abstract data type
- Primitive data type
- Non primitive data type
- Linear DS vs non linear DS
- Array

Definition of Data structure

- A **Data Structure** is a way of organizing and storing data in a computer so that it can be accessed and modified efficiently. It defines the relationship between data elements and the operations that can be performed on them.
- **Examples:** Arrays, Linked Lists, Stacks, Queues, Trees, Graphs, etc.

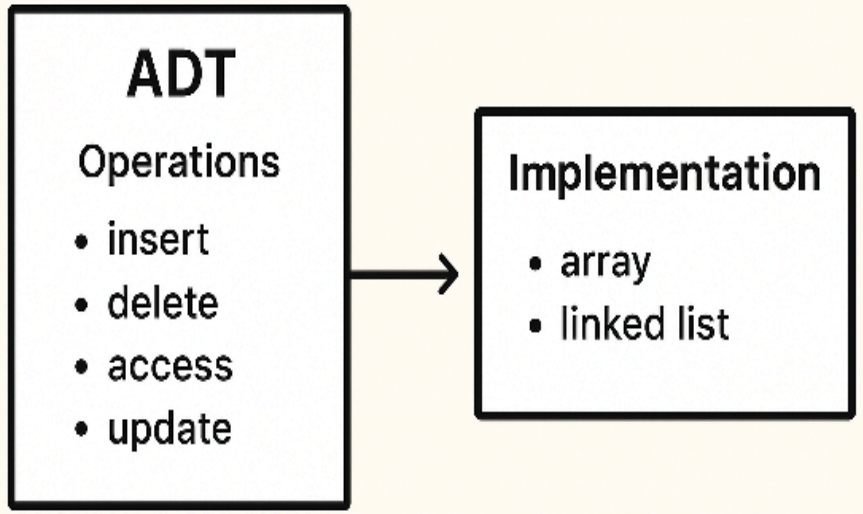
```
Array Example (Linear Structure):
Index: 0       1       2
Data:  Alice   Bob     Carol

Linked List Example (Non-Linear Structure):
[Alice] -> [Bob] -> [Carol] -> NULL

Tree Example (Hierarchical Structure):
        Alice
       /     \
     Bob     Carol
```

Abstract Data Type (ADT)

- An **Abstract Data Type (ADT)** is a **mathematical model** for a certain class of data structures that **specifies the behavior** (operations) of the data type **without specifying how it is implemented**.
- It focuses on **what operations are to be performed** rather than **how these operations are implemented**.
- ADT provides a **logical description** of the data and the operations on it.
- Examples: **Stack, Queue, List, Deque, Priority Queue, Set**

Abstract Data Type (ADT) cont…

- ADT is a **conceptual model**; it hides the implementation details.
- Operations include:
  - **Insertion**
  - **Deletion**
  - **Access/Retrieval**
  - **Update/Modification**
- Implementation can vary:
  - **Using arrays**
  - **Using linked lists**
  - **Using other data structures**



Common Examples of ADTs(Stack)

- **Definition**: A stack is a collection where elements are added and removed in a **Last In, First Out (LIFO)** order.
- **Operations**:
  1. `push(x)` → Add element `x` to the top.
  2. `pop()` → Remove and return the top element.
  3. `peek()` → Return the top element without removing it.
  4. `isEmpty()` → Check if the stack is empty.

- **Implementation**: Can be done using an array or linked list.

Common Examples of ADTs(Queue)

- **Definition**: A queue is a collection where elements are added at the **rear** and removed from the **front** (**First In, First Out - FIFO**).
- **Operations**:

1. `enqueue(x)` → Add element `x` at the rear.
2. `dequeue()` → Remove and return element from the front.
3. `front()` → Get the front element without removing it.
4. `isEmpty()` → Check if the queue is empty.

- Example :People waiting in a line → first person to come is first person served.

Primitive Data Types

- Primitive data types are the **basic building blocks** provided by a programming language. They are **simple** and **predefined**.
- **Characteristics:**
  - Stores **single values**.
  - **Efficient** in memory usage.
  - Predefined by the language.

Examples:

Data Type	Description	Example
int	Stores integers	`int age = 25;`
float	Stores decimal numbers	`float price = 19.99;`
char	Stores a single character	`char grade = 'A';`
boolean	Stores true/false	`boolean isActive = true;`
double	Stores larger decimal numbers	`double pi = 3.14159;`

Non-Primitive Data Types

- Non-primitive data types are **more complex**, usually created by the programmer or provided by libraries. They can store **multiple values** or more **complex data structures**.
- **Characteristics:**
  - Not predefined (except some like String in Java).
  - Can store **collections of values**.
  - Can be **manipulated with methods/functions**.
  - Typically **reference types** (they store memory addresses instead of actual values).

Non-Primitive Data Types cont..

Examples:

Data Type	Description	Example
String	Stores text	`String name = "Ahmed";`
Arrays	Stores multiple values of same type	`int[] numbers = {1,2,3};`
Classes/Objects	Custom data types	`Car myCar = new Car();`
Lists (Java)/ArrayList	Dynamic collection of elements	`ArrayList<Integer> list = new ArrayList<>();`

Feature	Primitive Data Type	Non-Primitive Data Type
Definition	Basic, predefined	Complex, user-defined or library-based
Size	Fixed (depends on type)	Dynamic / depends on content
Memory	Stored directly	Stored as reference
Examples	int, float, char, boolean	String, Array, Object
Methods	No built-in methods	Can have built-in methods

Linear Data Structures (LDS)

- A linear data structure is a structure in which elements are **arranged sequentially**, and each element is **connected to its previous and next element** (except the first and last).
- **Characteristics:**
  - Elements are stored **in order**.
  - **Single level** organization.
  - Traversal is **simple** (one after another).
  - Memory can be **contiguous** (like arrays) or **linked** (like linked lists).

Examples:

Structure	Description	Example
Array	Collection of elements of the same type	`int arr[] = {1,2,3,4};`
Linked List	Elements linked using pointers	`Node1 -> Node2 -> Node3`
Stack	LIFO (Last In First Out)	Push/pop operations
Queue	FIFO (First In First Out)	Enqueue/dequeue operations

Non-Linear Data Structures (NLDS)

- A non-linear data structure is a structure in which elements are **not arranged sequentially**. **Each element can connect to multiple elements**, forming a **hierarchy or network**.
- **Characteristics:**
  - Elements are **not in order**.
  - **Multi-level** organization.
  - Traversal requires **more complex algorithms** (like recursion, DFS, BFS).
  - Memory is usually **dynamic**.

Examples:

Structure	Description	Example
Tree	Hierarchical structure (root → children)	Binary Tree, Binary Search Tree
Graph	Network of nodes with edges	Social networks, Google Maps connections

Key Differences:

Feature	Linear DS	Non-Linear DS
Arrangement	Sequential	Hierarchical / Network
Levels	Single level	Multiple levels
Traversal	Easy, one by one	Complex, needs recursion or algorithms
Memory usage	Contiguous or linked	Usually dynamic
Examples	Array, Linked List, Stack, Queue	Tree, Graph

Array

- An **array** is a **collection of elements of the same data type** stored in **contiguous memory locations**.
- All elements are accessed using an **index**.
- Indexing **starts from 0** in C++.
- Syntax:

```
data_type array_name[size];
```

- Example :

```
int numbers[5]; // Declares an array of 5 integers
```

Array

- Initializing an Array:

You can assign values at the time of declaration:

```cpp
int numbers[5] = {10, 20, 30, 40, 50};
```

Or assign values later:

```cpp
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
```

Array

- Accessing Array Elements

```cpp
#include <iostream>
using namespace std;

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    cout << "First element: " << numbers[0] << endl; // 10
    cout << "Third element: " << numbers[2] << endl; // 30

    numbers[1] = 25; // Modify second element
    cout << "Modified second element: " << numbers[1] << endl; // 25

    return 0;
}
```

```
First element: 10
Third element: 30
Modified second element: 25
```

Array

- Traversing an Array

```
#include <iostream>
using namespace std;

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    cout << "Array elements: ";
    for(int i = 0; i < 5; i++) {
        cout << numbers[i] << " ";
    }
    cout << endl;

    return 0;
}
```

```
Array elements: 10 20 30 40 50
```