UNIT II – LIST, STACK AND QUEUE

Data Structures - Abstract Data Types (ADT) - List ADT and Array Implementation - Linked List - Doubly Linked List - Circular Linked List - Array and Linked List implementation of Stack - Application: Infix to Postfix Conversion - Postfix Expression Evaluation – Array and Linked List implementation of Queues –Applications in signal processing pipelines, interrupt handling, UART buffer design.

INTRODUCTION

2.1 Data Structures Definition

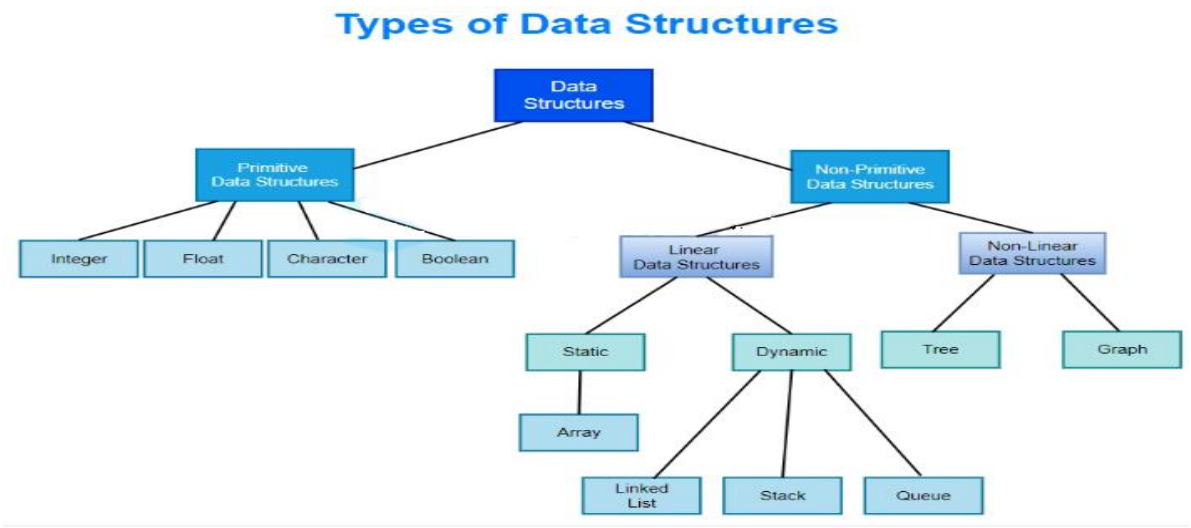
Data Structure is a way of organizing and retrieving data in such a way that it can be accessed, modified or perform operations on these data in an effective way.

Applications of Data Structures

- Compiler design
- Operating system
- Statistical analysis package
- DBMS
- Numerical analysis
- Simulation
- Artificial Intelligence

CLASSIFICATION OF DATA STRUCTURES

Data structures are generally categorized into two classes: primitive and non-primitive data structures.



Primitive data structures are the fundamental data types which are supported by a programming language. Some basic(built-in) data types are integer, real, character, and Boolean.

Non-primitive data structures are those data structures which are created using primitive(User-Defined) data structures. Examples of such data structures include linked lists, stacks, trees, and graphs. Non-primitive data structures can further be classified into two categories: linear and non-linear data structures.

LINEAR AND NON-LINEAR STRUCTURES

LINEAR DATA STRUCTURES

If the elements of a data structure are stored in a linear or sequential order, then it is a linear data structure. Examples include arrays, linked lists, stacks, and queues.

- **Static data structure:** Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
  Example: array data structure.
- **Dynamic data structure:** In the dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.
  Examples: stack and queue data structures.

NON-LINEAR DATA STRUCTURES

If the elements of a data structure are not stored in a sequential order, then it is a non-linear data structure. The relationship of adjacency is not maintained between elements of a non-linear data structure. In a non-linear data structure, we can't traverse all the elements in a single run. Examples include trees and graphs.

OPERATIONS ON DATA STRUCTURES

Traversal: Visit every part of the data structure.

Search: Traversal through the data structure for a given element.

Insertion: Adding new elements to the data structure.
Deletion: Removing an element from the data structure.

Sorting: Rearranging the elements in some type of order (e.g Increasing or Decreasing).

Merging: Combining two similar data structures into one.

2.2 ABSTRACT DATA TYPE (ADT):

An **Abstract Data Type (ADT)** is a **conceptual or logical model** that defines a data structure in terms of:

- **The type of data to be stored**
- **The set of operations that can be performed on the data**
- **The behavior of these operations**, independent of how the data is stored or implemented in memory

An ADT specifies **what operations are performed**, not **how they are implemented**, thereby providing **data abstraction**.

Examples of ADT:

Stack, Queue, List, Set, and Map.

ADT focuses on **what operations are performed**, not **how they are implemented**

- It provides **data abstraction** by hiding internal details
- It is **independent of programming language**
- It allows **multiple implementations** for the same ADT
- It improves **modularity and clarity** in program design
- It separates **interface from implementation**
- ADTs help in **designing efficient and reusable programs**
- Common ADTs include **Stack, Queue, List, Set, and Map**
- ADTs form the **foundation of data structures**
- They make programs **easier to understand, maintain, and modify**