Introduction to Data Structure

Data structure is a representation of logical relationship existing between individual elements of data. In other words, a data structure defines a way of organizing all data items that considers not only the elements stored but also their relationship to each other. The term data structure is used to describe the way data is stored.

To develop a program of an algorithm we should select an appropriate data structure for that algorithm. Therefore, data structure is represented as:

Algorithm + Data structure = Program

A data structure is said to be **linear** if its elements form a sequence or a linear list. The linear data structures like an array, stacks, queues and linked lists organize data in linear order. A data structure is said to be **non linear** if its elements form a hierarchical classification where, data items appear at various levels.

Trees and **Graphs** are widely used **non-linear data structures**. Tree and graph structures represents hierarchial relationship between individual data elements. Graphs are nothing but trees with certain restrictions removed.

Importance of Data Structure

Due to the complexity of applications and the daily growth in data, there may be issues with processing **speed, data searching, handling numerous requests,** etc. Data structure offers a method for effectively **managing, organising, and storing data.** Data structures make it simple to navigate through the data elements. Data structures offer **productivity, reuse, and abstraction**. Because storing and retrieving user data as quickly as feasible is a program's primary job, it plays a significant role in improving performance.

Types of Data Structure

Data structures are divided into two types:

- **Primitive data structures.**
- **Non-primitive data structures.**

➔ Primitive Data Structure :

Primitive Data Structures are the basic data structures that directly operate upon the machine instructions. They have different representations on different computers. Integers, floating point numbers, character constants, string constants and pointers come under this category.

➔ Non-Primitive Data Structure :

Non-primitive data structures are more complicated data structures and are derived from primitive data structures. They emphasize on grouping same or different data items with relationship between each data item. Arrays, lists and files come under this category. Figure shows the classification of data structures.

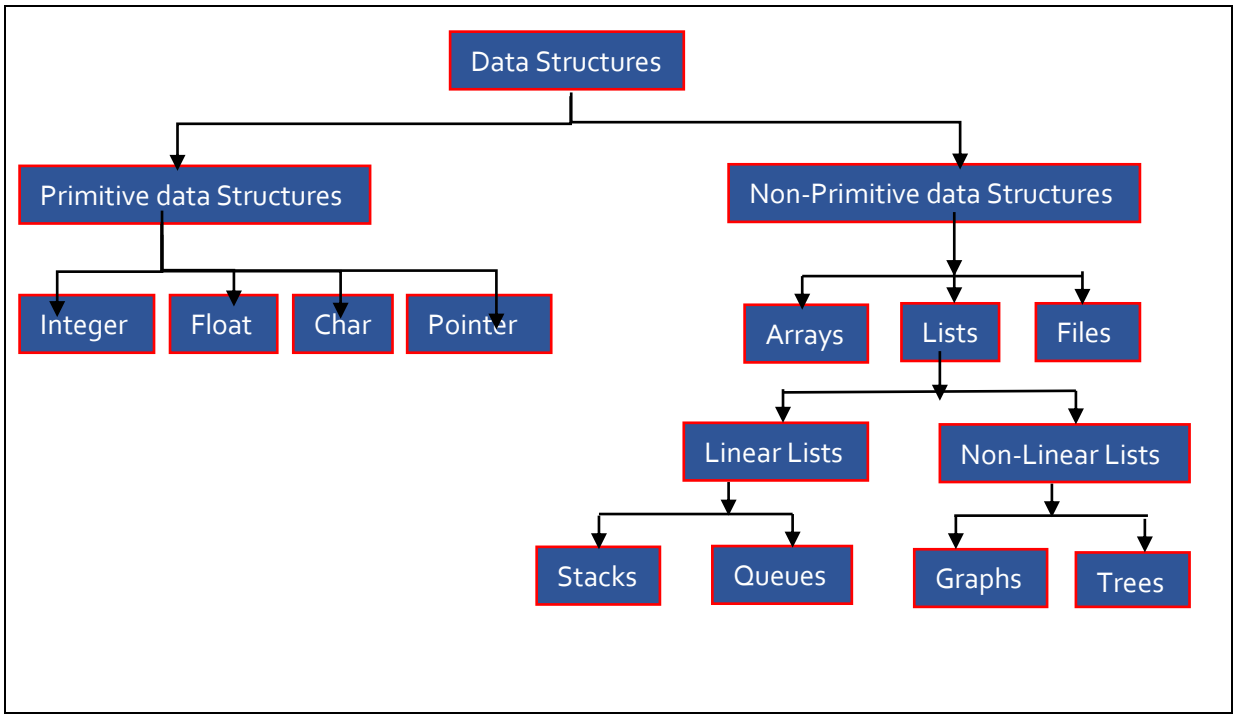


Figure : Classification of Data Structures.

The non-primitive data structure is divided into two types:

- **Linear data structure**
- **Non-linear data structure**

➔ Linear Data Structure

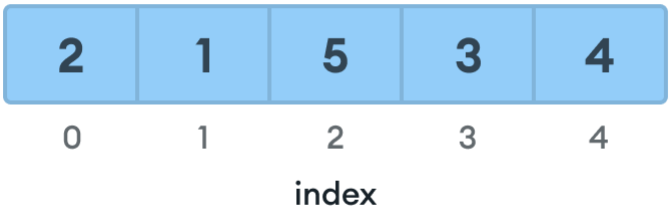
In **linear data structures**, the elements are arranged in sequence one after the other.

Elements are arranged in particular order, they are easy to implement.

Popular linear data structures are:

1. Array Data Structure

In an array, elements in memory are arranged in continuous memory. All the elements of an array are of the same type. And, the type of elements that can be stored in the form of arrays is determined by the programming language.

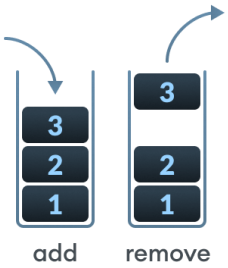


An array with each element represented by an index

2. Stack Data Structure

In stack data structure, elements are stored in the LIFO principle. That is, the last element stored in a stack will be removed first.

It works just like a pile of plates where the last plate kept on the pile will be removed first.



In a stack, operations can be perform only from one end (top here).

3. Queue Data Structure

Unlike stack, the queue data structure works in the FIFO principle where first element stored in the queue will be removed first.

It works just like a queue of people in the ticket counter where first person on the queue will get the ticket first.

In a queue, addition and removal are performed from separate ends.

4. Linked List Data Structure

In linked list data structure, data elements are connected through a series of nodes. And, each node contains the data items and address to the next node.

A linked list

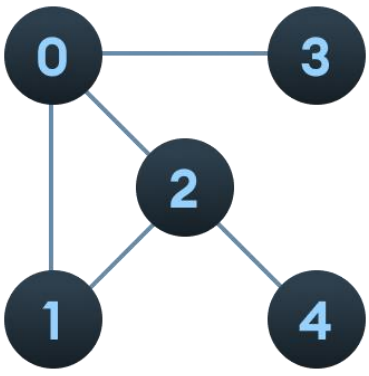
➔ Non-Linear Data Structure

Unlike linear data structures, elements in non-linear data structures are not in any sequence. Instead they are arranged in a hierarchical manner where one element will be connected to one or more elements.

Non-linear data structures are further divided into graph and tree based data structures.

1. Graph Data Structure

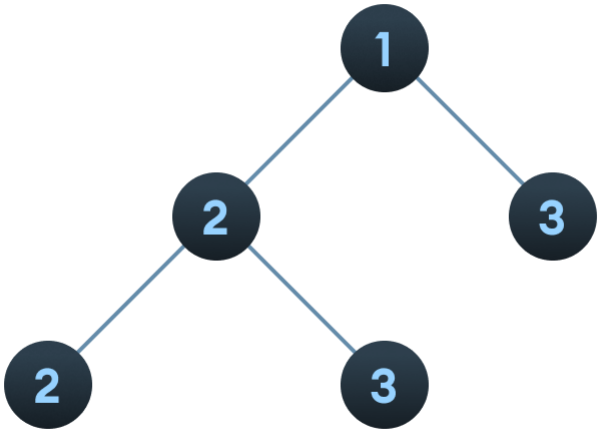
In graph data structure, each node is called vertex and each vertex is connected to other vertices through edges.



Graph data structure example

2. Trees Data Structure

Similar to a graph, a tree is also a collection of vertices and edges. However, in tree data structure, there can only be one edge between two vertices.



Tree data structure example

➔ Linear Vs Non-linear Data Structures

Now that we know about linear and non-linear data structures, let's see the major differences between them.

Linear Data Structures	Non Linear Data Structures
• The data items are arranged in sequential order, one after the other.	The data items are arranged in non-sequential order (hierarchical manner).
• All the items are present on the single layer.	The data items are present at different layers.
• It can be traversed on a single run. That is, if we start from the first element, we can traverse all the elements sequentially in a single pass.	It requires multiple runs. That is, if we start from the first element it might not be possible to traverse all the elements in a single pass.

• The memory utilization is not efficient.	Different structures utilize memory in different efficient ways depending on the need.
• The time complexity increase with the data size.	Time complexity remains the same.
• Example: Arrays, Stack, Queue	Example: Tree, Graph, Map

Data structures: Organization of data

The collection of data you work with in a program have some kind of structure or organization. No matter how complex your data structures are they can be broken down into two fundamental types:

- **Contiguous**
- **Non-Contiguous**

➔Contiguous Structures

In **contiguous structures**, terms of data are kept together in memory (either RAM or in a file). An **array** is an example of a contiguous structure. Since each element in the array is located next to one or two other elements.

➔Non-Contiguous Structures

Items in a **non-contiguous structure** and scattered in memory, but we linked to each other in some way. A **linked list** is an example of a non-contiguous data structure. Here, the nodes of the list are linked together using pointers stored in each node.

Figure below illustrates the difference between contiguous and non-contiguous structures.

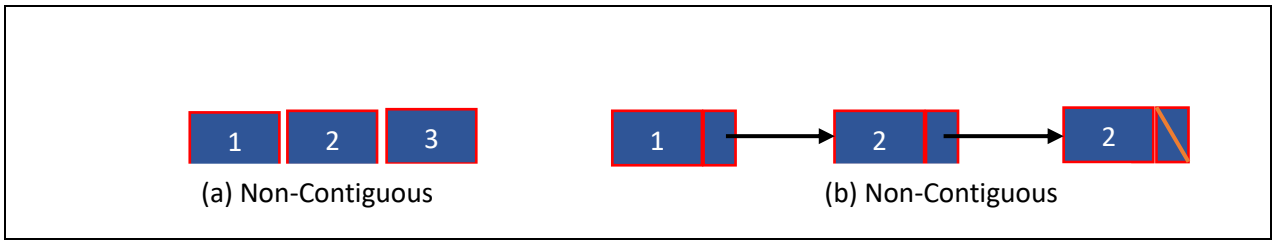


Figure : Contiguous and Non-contiguous structures compared

➔Hybrid structures

If two basic types of structures are mixed then it is a hybrid form. Then one part contiguous and another part non-contiguous.

For example, figure shows how to implement a double–linked list using three parallel arrays, possibly stored a past from each other in memory.

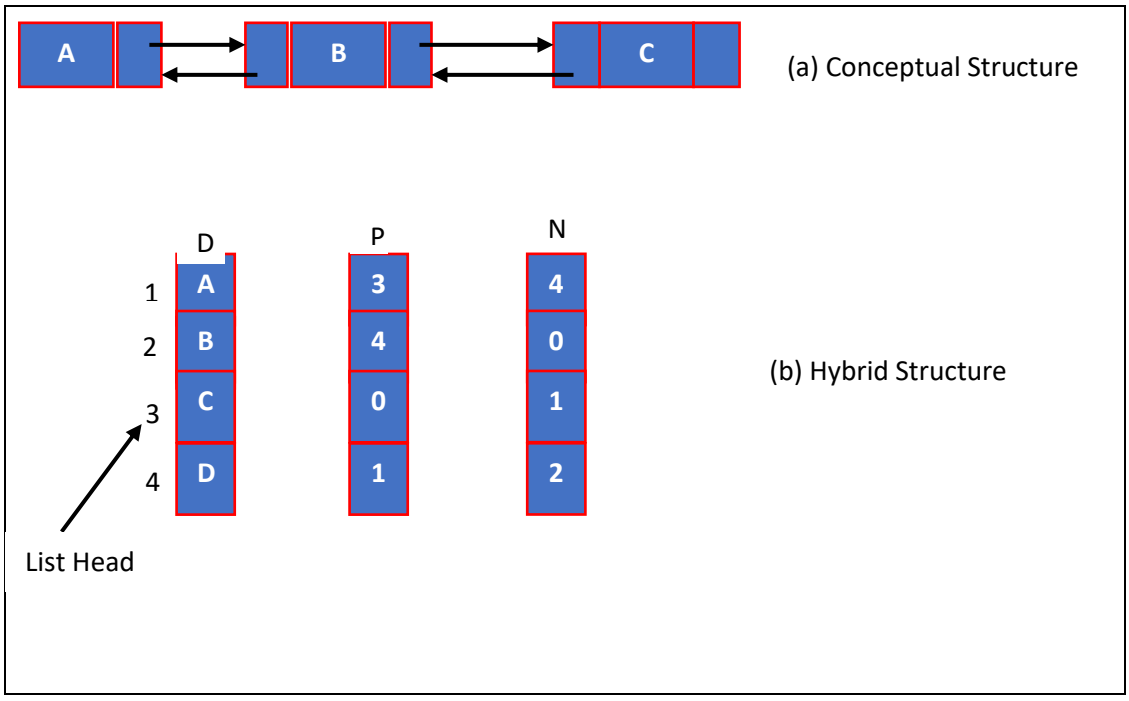


Figure : A double linked list via a hybrid data structure

The array D contains the data for the list, whereas the array P and N hold the previous and next "pointers". The pointers are actually nothing more than indexes into the D array. For instance, D[i] holds the data for node i and p[i] holds the index to the node previous to i, where may or may not reside at position i–1. Like wise, N[i] holds the index to the next node in the list.

Operation of Data Structure

Data structure operations are the methods used to manipulate the data in a data structure. The most common data structure operations are:

- **Traversal :** Traversal operations are used to visit each node in a data structure in a specific order. This technique is typically employed for **printing**, **searching**, **displaying**, and **reading** the data stored in a data structure.
- **Insertion :** Insertion operations add **new data elements** to a data structure. You can do this at the data structure's beginning, middle, or end.
- **Deletion :** Deletion operations remove data elements from a data structure. These operations are typically performed on nodes that are no longer needed.
- **Search :** Search operations are used to find a **specific data element** in a data structure. These operations typically employ a compare function to determine if two data elements are equal.
- **Sort :** Sort operations are used to **arrange** the data elements in a data structure in a specific order. This can be done using various sorting algorithms, such as insertion sort, bubble sort, merge sort, and quick sort.
- **Merge :** Merge operations are used to **combine** two data structures into one. This operation is typically used when two data structures need to be combined into a single structure.

- **Copy :** Copy operations are used to create a **duplicate** of a data structure. This can be done by copying each element in the original data structure to the new one.

Application of Data Structure

Data structures have many applications, such as:

➔ **Data Storage:**

Data structures facilitate efficient data persistence, like specifying attribute collections and corresponding structures used in database management systems to store records.

➔ **Data Exchange:**

Organized information, defined by data structures, can be shared between applications like TCP/IP packets.

➔ **Resource and Service Management:**

Data structures such as linked lists can enable core operating systems resources and services to perform functions like file directory management, memory allocation, and processing scheduling queues.

➔ **Scalability:**

Big data applications rely on data structures to manage and allocate data storage across many distributed storage locations. This function guarantees scalability and high performance.

Advantages of Data structures

- Data structure facilitates effective data storage in storage devices.
- The use of data structures makes it easier to retrieve data from a storage device.
- The data structure allows for the effective and efficient processing of both little and big amounts of data.
- Manipulation of vast amounts of data is simple when a proper data structure technique is used.
- The use of a good data structure may assist a programmer to save a lot of time or processing time while performing tasks such as data storage, retrieval, or processing.
- Most well-organized data structures, including stacks, arrays, graphs, queues, trees, and linked lists, have well-built and pre-planned approaches for operations such as storage, addition, retrieval, modification, and deletion. The programmer may totally rely on these facts while utilising them.
- Data structures such as arrays, trees, linked lists, stacks, graphs, and so on are thoroughly verified and proved, so anybody may use them directly without the need for study and development. If you opt to design your own data structure, you may need to do some study, but it will almost certainly be to answer a problem that is more sophisticated than what these can supply.
- In the long term, data structure utilization might merely encourage reusability.