# Introduction to Data Structure & Algorithm

Charles Aunkan Gomes
Lecturer, Dept. of CSE
United International University

# Good Computer Program

A computer program is a **series of instructions** to carry out a particular task written in a language that a can computer can understand.

A good computer program consists of two things: **data structure & algorithm**

A good program uses the best data structure and algorithm for the task, making it fast, efficient, and easy to maintain.

Think of a class lineup of students standing in random order. To sort them by height, you go down the line, swapping students if they're out of order until everyone is arranged from shortest to tallest.

## Algorithm

An algorithm is a step-by-step procedure for solving an instance of a problem  in finite number of steps.

For example: the process of preparing a cup of coffee or tea.

- Fill the kettle with water
- Boil the water in the kettle
- Put teabag in a cup
- Pour boiled water into the cup
- Add sugar to the cup
- Stir the cup
- Drink the tea

## Algorithm

An algorithm must contain the following properties:

- **Input**: An algorithm must take zero or inputs.
- **Output**: An algorithm must produce one or more outputs.
- **Finiteness**: The algorithm must terminate after a finite number of steps.
- **Definiteness**: All steps of the algorithm must be precisely defined. Every instruction should be clear and unambiguous.
- **Effectiveness**: Algorithm should be capable of producing correct output for all possible valid inputs within a finite amount of time.

## Data Structure

**Data** is a collection of facts such as numbers, words, measurements etc.

**Data structure** is the representation of the logical relationship existing between individual elements of data.

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

In computer science and computer programming, a data structure might be selected or designed to store data for the purpose of using it with various algorithms -- commonly referred to as data structures and algorithms (DSA).

# Data Structure

Sometimes a QUEUE makes sense...
Quack add!
quack remove!
...sometimes just a SET is enough.
quack remove!
Eventually a MAP is necessary...
rick
nick
bob
cris
pingo
juan
quack! put
...but sooner or later we all need to deal with a STACK.
BILLS
POP!

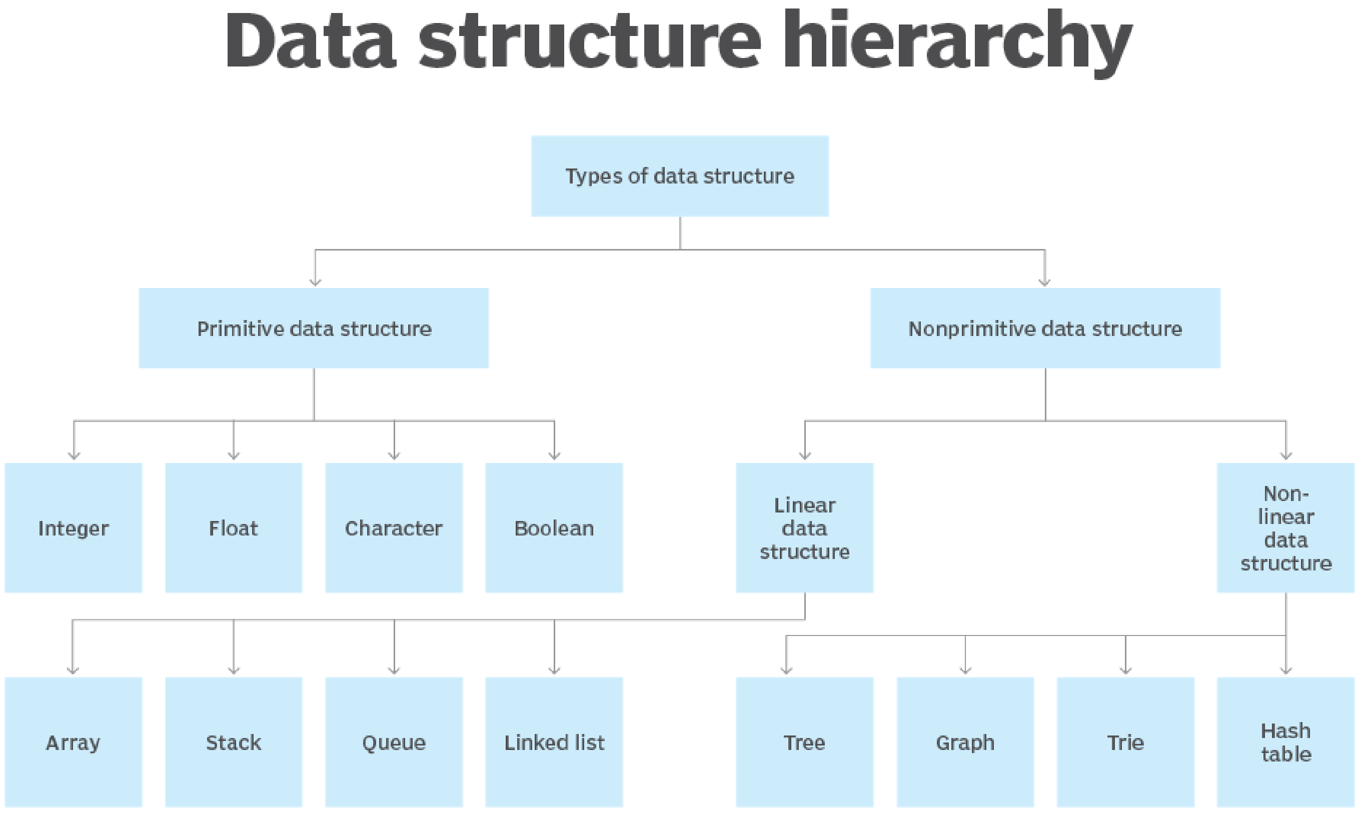
## Why we need Data Structure?

Think of a dictionary. When you want to find the meaning of a word, you don’t read the dictionary from start to finish. Instead, you go directly to the section where the word is likely to be, based on its first letter or a group of letters.

Now, think of searching a word in a scrambled dictionary……..BAM!!

Data structures are fundamental tools that software developers use to organize and store data efficiently. The choice of appropriate data structure can make the difference between running a program in a few seconds or many days.

# Classification of Data Structure



## Primitive Data Structure

**Primitive data structures** are the most basic types of data provided directly by a programming language. These types are often simple, with a single piece of data, and are directly supported at the machine level.

Primitive data types are used to perform simple operations, such as arithmetic, logical comparisons, and basic storage of values.

Examples:

**Integer**: Represents whole numbers, e.g., 5, -3, 0.

**Float/Double**: Represents decimal numbers, e.g., 3.14, -0.001.

**Character**: Represents a single character, e.g., 'a', 'Z’.

**Boolean**: Represents logical values, true or false.

**What about Pointers!!!**

## Non-Primitive Data Structure

**Non-primitive data structures** are derived from primitive types and can store multiple values or complex data. They are designed to hold large amounts of data and to manage it in a structured way, often involving **heterogenous data types** or **hierarchical arrangements.**

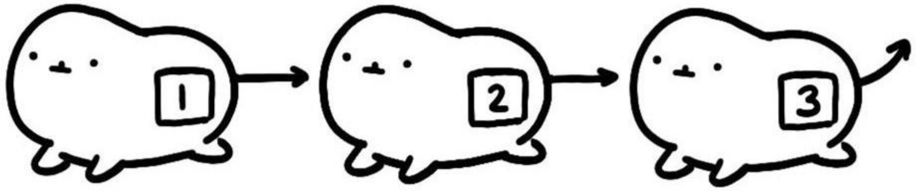
**Composite**: Can hold multiple values, and each element can be a different type (in heterogeneous cases).

**Complex Operations**: Support complex operations for insertion, deletion, sorting, and searching.

## Linear Data Structure

Non-primitive data structures can be further divided into two categories:

- Linear Data Structure: Data elements are arranged **sequentially**, with each element connected directly to the next and/or previous element. Memory allocation is often **contiguous**, which makes access predictable. Operations (like traversal) are straightforward due to sequential arrangement.



## Array

Arrays are contiguous memory locations that store elements of the same type. The primary advantage of arrays is the ability to access any element directly using its index. This direct access provides constant time complexity for retrieval operations. However, arrays have a fixed size, which means resizing them can be a challenge.

**Example**: Consider a scenario where you have **a list of student names** in a classroom. If you know a student's roll number, you can directly access their name from the list without checking each name one by one.

**Real life applications:** Database indexing, Image processing etc.

## Linked List

Unlike arrays, linked lists are a collection of nodes where each node contains a data element and a reference to the next node in the sequence. This structure allows for dynamic size adjustments, making operations like insertion and deletion simpler compared to arrays. However, accessing a specific element might require traversing from the start to the desired node.

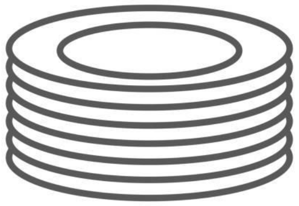
**Example**: Think of a train, where each compartment represents a node. To reach the last compartment, you'd have to pass through each one sequentially.

**Real life applications:** File system directory, Browser navigation etc.

## Stack

Stacks follow the **Last-In-First-Out (LIFO)** principle, meaning the last item added is the first one to be removed. It can also be stated as **First-In-Last-Out (FILO)** where the item added first is the last one to be removed.

**Example**: Imagine a stack of plates at a buffet. You always add new plates on top and take plates from the top. To reach a plate at the bottom, you'd have to remove all the plates above it.

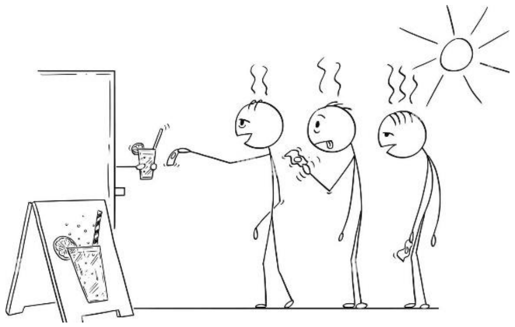


**Real life applications:** Function calls, Browser history, Undo feature etc.

## Queue

Queues follow the **First-In-First-Out (FIFO)** principle, meaning the first item added is the first one to be removed. It can also be stated as **Last-In-Last-Out (LILO)** where the item added last is the last one to be removed.

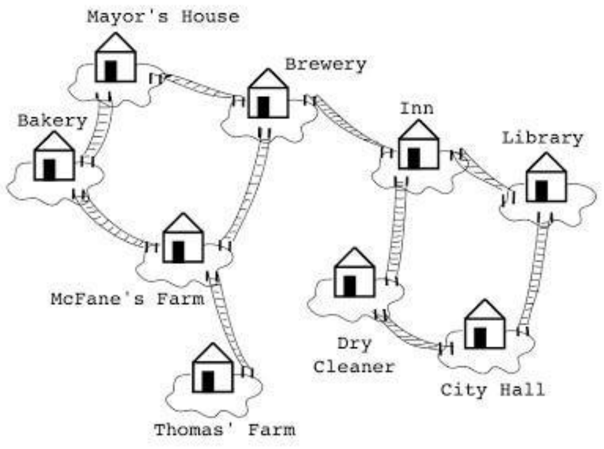
**Example**: Think of a line at a juice shop. The first person in line is served first, and new people join at the end of the line. You can only serve people from the front.



**Real life applications:** Ticket counter, Printer queue, Operating system's task scheduling etc.

# Non-linear Data Structure

Non-linear Data Structure: Data elements are arranged **hierarchically** or in a **more complex relationship**, without a strict sequence. Memory may not be contiguous. Operations (like search or traversal) can be complex but are suitable for relationships not easily represented linearly.



# Graph

A graph is a collection of **points (nodes)** connected by **lines (edges)** that may form a web of connections.

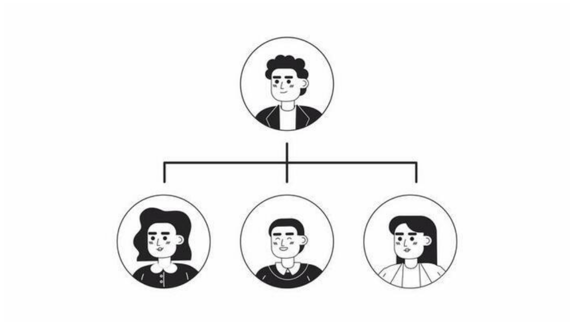
**Example:** Imagine a **social network** where each person represents a node, and each friendship is a connection (edge) between two people. You can have multiple paths between friends, and some people may be connected to many others, while some have only a few connections.

**Real life applications:** Graphs are used for complex relationships, like social networks or city maps.

## Tree

A tree is a **hierarchical structure** with a **root** element branching out to **child** elements, much like a family tree.

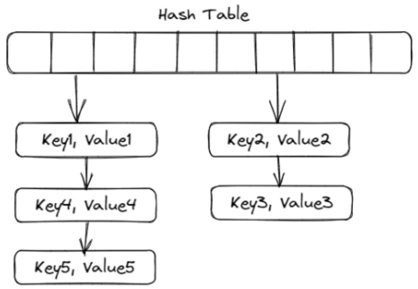
**Example:** Imagine a **company's organization chart**: the CEO is at the top (root), with managers below, and employees under each manager. You can follow a clear path from the CEO down to any employee.



**Real life applications:** Trees are useful for showing hierarchical relationships, like organizational charts or file systems.

# Hash Table

A hash table stores data in **key-value pairs** and allows for very fast data retrieval. A hash table uses a **hash function** to map keys to specific locations (indexes) in an array, allowing for quick access to values. When two keys map to the same location (a collision), it handles this with methods like **chaining** or **open addressing**.



**Real life applications:** Database indexing, Caching, Dictionaries etc.

## Thank You