1 Introduction

Objectives

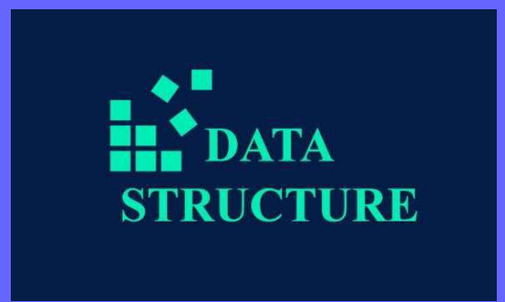
After completing this lesson, you should be able to:

- Introduction to Data Structures
- Objects and Collections
- Constructors and Destructors
- Types of Data Structures (Arrays …)

Course Roadmap

Data Structures

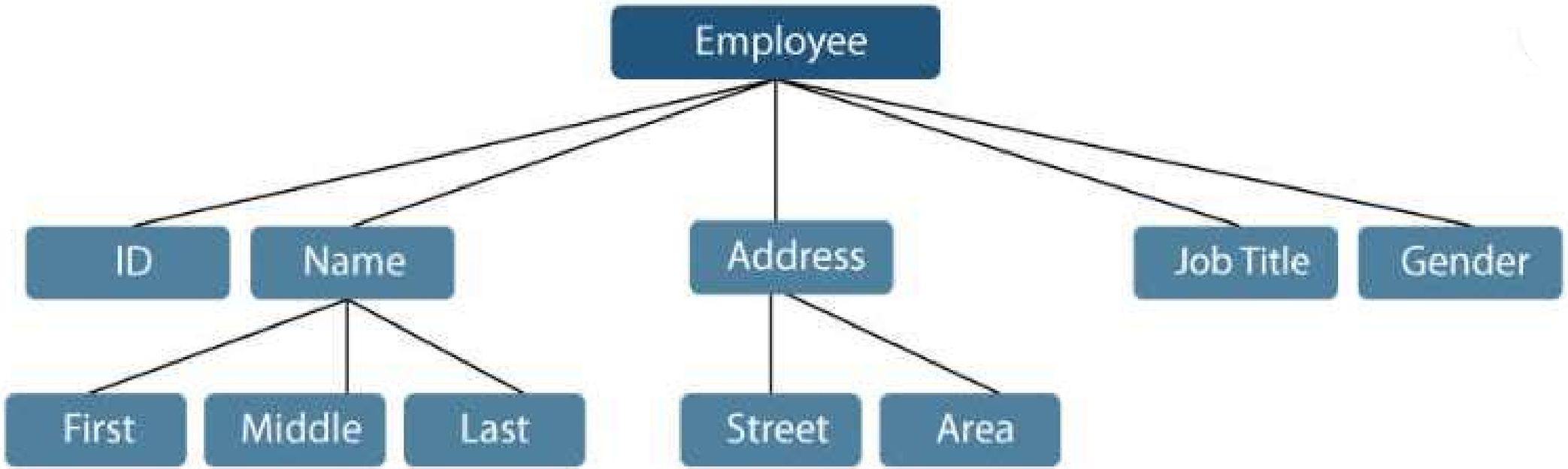
- **Lesson 1: Introduction to Data Structures** — You are here!
- Lesson 2: Lists and Stacks
- Lesson 3: Searching
- Lesson 4: Trees and Queues
- Lesson 5: Sorting



Introduction to Data Structures

Introduction

- "**Data**" to refer to Computer Information, either transmitted or stored. However, there is data that exists in order types as well. Data can be numbers or texts written on a piece of paper, in the form of bits and bytes stored inside the memory of electronic devices, or facts stored within a person's mind.
- As the world started modernizing, this data became a significant aspect of everyone's day-to-day life, and various implementations allowed them to store it differently.
- **Data** is a collection of facts and figures or a set of values or values of a specific format that refers to a single set of item values. The data items are then classified into sub-items, which is the group of items that are not known as the simple primary form of the item.


Employee
ID
Name
Address
Job Title
Gender
First
Middle
Last
Street
Area

Introduction

- *A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.*

- A data structure is not only used for organizing the data. It is also used for processing, retrieving, and storing data. There are different basic and advanced types of data structures that are used in almost every program or software system that has been developed.

What is Data Structure?

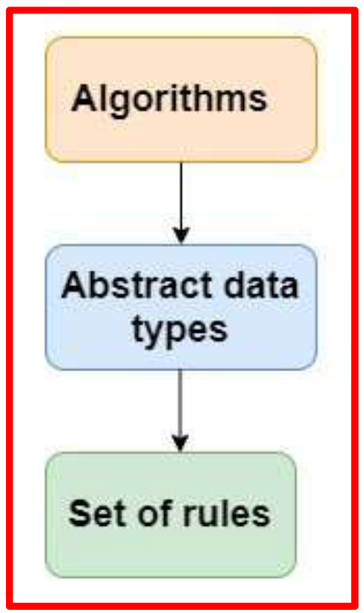
- The data structure name indicates itself that organizing the data in memory. There are many ways of organizing the data in the memory as we have already seen one of the data structures, i.e., array in C language.

- Array is a collection of memory elements in which data is stored sequentially, i.e., one after another. In other words, we can say that array stores the elements in a continuous manner.

- This organization of data is done with the help of an array of data structures. There are also other ways to organize the data in memory.

Cont …

- The data structure is not any programming language like C, C++, java, etc. It is a set of algorithms that we can use in any programming language to structure the data in the memory.
- To structure the data in memory, 'n' number of algorithms were proposed, and all these algorithms are known as Abstract data types. These abstract data types are the set of rules.



Basic Terminologies related to Data Structures

- Data Structures are the building blocks of any software or program. Selecting the suitable data structure for a program is an extremely challenging task for a programmer.

1. **Data:** We can define data as an elementary value or a collection of values. For example, the Employee's name and ID are the data related to the Employee.
2. **Data Items:** A Single unit of value is known as Data Item.
3. **Group Items:** Data Items that have subordinate data items are known as Group Items. For example, an employee's name can have a first, middle, and last name.
4. **Elementary Items:** Data Items that are unable to divide into sub-items are known as Elementary Items. For example, the ID of an Employee.
5. **Entity and Attribute:** A class of certain objects is represented by an Entity. It consists of different Attributes. Each Attribute symbolizes the specific property of that Entity. For example,

The term "information" is sometimes utilized for data with given attributes of meaningful or processed data.

1. **Field:** A single elementary unit of information symbolizing the Attribute of an Entity is known as Field.
2. **Record:** A collection of different data items are known as a Record. For example, if we talk about the employee entity, then its name, id, address, and job title can be grouped to form the record for the employee.
3. **File:** A collection of different Records of one entity type is known as a File. For example, if there are 100 employees, there will be 25 records in the related file containing data about each employee.

Why should we learn Data Structures?

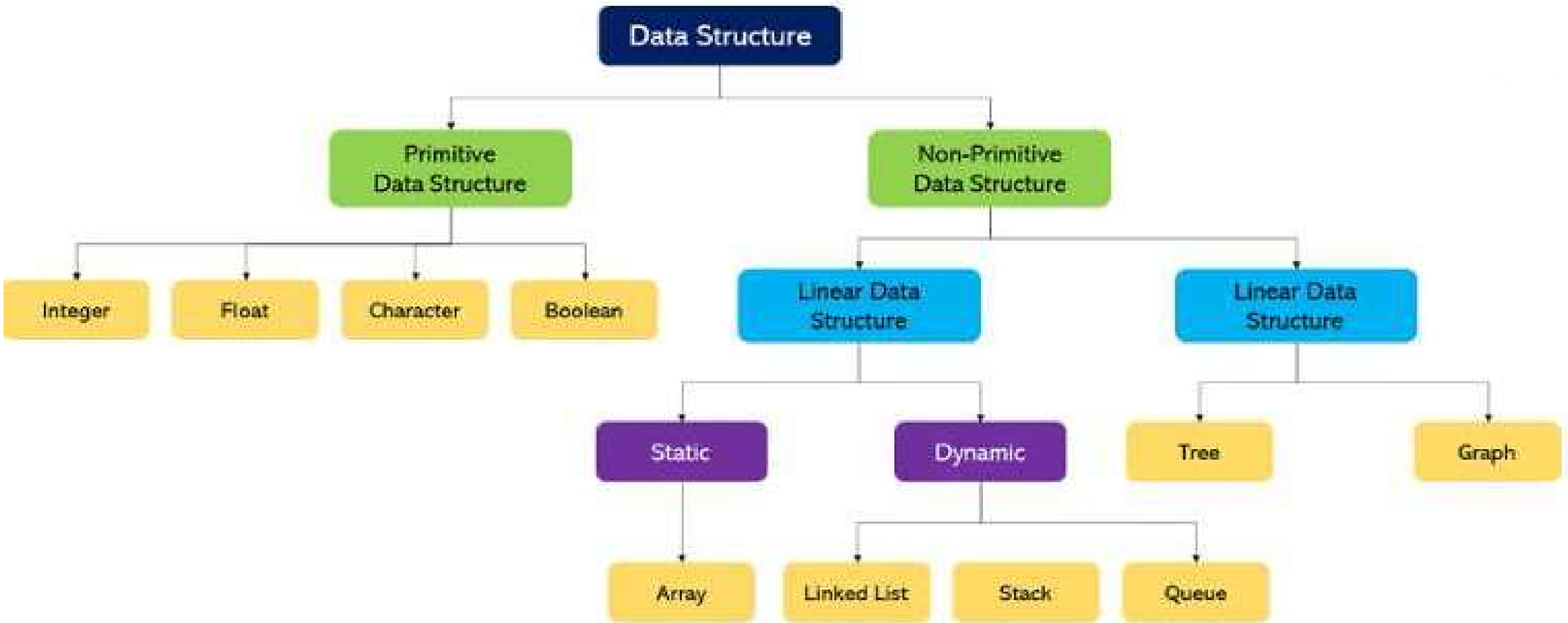
1. Data Structures and Algorithms are two of the key aspects of Computer Science.
2. Data Structures allow us to organize and store data, whereas Algorithms allow us to process that data meaningfully.
3. Learning Data Structures and Algorithms will help us become better Programmers.
4. We will be able to write code that is more effective and reliable.
5. We will also be able to solve problems more quickly and efficiently.

Understanding the Objectives of Data Structures

1. **Correctness:** Data Structures are designed to operate correctly for all kinds of inputs based on the domain of interest. In order words, correctness forms the primary objective of Data Structure, which always depends upon the problems that the Data Structure is meant to solve.
2. **Efficiency:** Data Structures also requires to be efficient. It should process the data quickly without utilizing many computer resources like memory space. In a real-time state, the efficiency of a data structure is a key factor in determining the success and failure of the process.

Types of DataStructures

- There are two types of data structures:
  1. Primitive data structure
  2. Non-primitive data structure


Data Structure
Primitive Data Structure
Non-Primitive Data Structure
Integer
Float
Character
Boolean
Linear Data Structure
Linear Data Structure
Static
Dynamic
Tree
Graph
Array
Linked List
Stack
Queue

Primitive Data Structures

1. **Primitive Data Structures** are the data structures consisting of the numbers and the characters that come **in-built** into programs.
2. These data structures can be manipulated or operated directly by machine-level instructions.
3. Basic data types like **Integer, Float, Character**, and **Boolean** come under the Primitive Data Structures.
4. These data types are also called **Simple data types**, as they contain characters that can't be divided further

Non-Primitive Data Structures

1. **Non-Primitive Data Structures** are those data structures derived from Primitive Data Structures.
2. These data structures can't be manipulated or operated directly by machine-level instructions.
3. The focus of these data structures is on forming a set of data elements that is either **homogeneous** (same data type) or **heterogeneous** (different data types).
4. Based on the structure and arrangement of data, we can divide these data structures into two sub-categories -
    1. Linear Data Structures
    2. Non-Linear Data Structures

Linear Data Structure

- The arrangement of data in a sequential manner is known as a linear data structure.
- The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues.
- In these data structures, one element is connected to only one another element in a linear form.
- However, it is not necessarily true in the case of memory, as the arrangement may not be sequential.

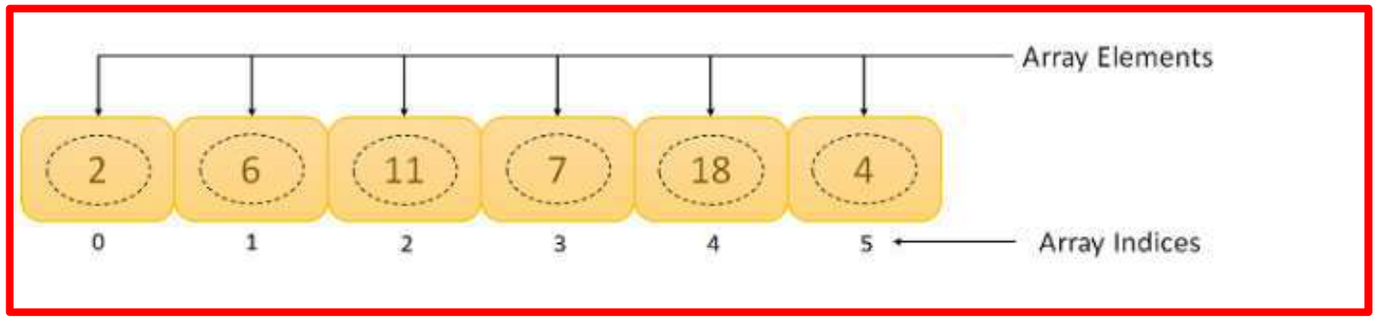
Further Classification

Data structures can also be classified as:

- **Static data structure:** It is a type of data structure where the size is allocated at the compile time. Therefore, the maximum size is fixed.
  - The **Array** is the best example of the Static Data Structure as they have a fixed size, and its data can be modified later.
- **Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible.
  - **Linked Lists, Stacks**, and **Queues** are common examples of dynamic data structures

1. Arrays

- An **Array** is a data structure used to collect multiple data elements of the same data type into one variable. Instead of storing multiple values of the same data types in separate variable names, we could store all of them together into one variable.
- An Array is a list of elements where each element has a unique place in the list. The data elements of the array share the same variable name; however, each carries a different index number called a subscript.
- We can access any data element from the list with the help of its location in the list. Thus, the key feature of the arrays to understand is that the data is stored in contiguous memory locations, making it possible for the users to traverse through the data elements of the array using their respective indexes.



Arrays can be classified into different types:

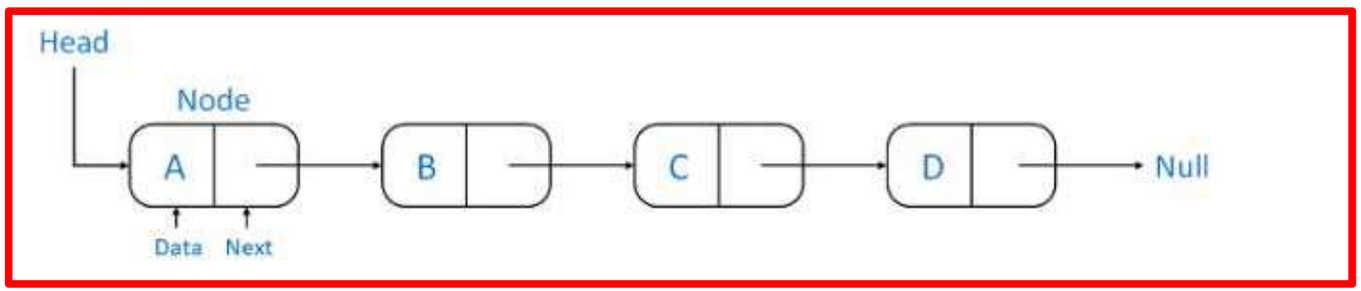
- **One-Dimensional Array:** An Array with only one row of data elements is known as a One-Dimensional Array. It is stored in ascending storage location.
- **Two-Dimensional Array:** An Array consisting of multiple rows and columns of data elements is called a Two-Dimensional Array. It is also known as a Matrix.

Some Applications of Array:

1. We can store a list of data elements belonging to the same data type.
2. Array acts as an auxiliary storage for other data structures.
3. The array also helps store data elements of a binary tree of the fixed count.
4. Array also acts as a storage of matrices.

2. Linked Lists

- A **Linked List** is another example of a linear data structure used to store a collection of data elements dynamically. Data elements in this data structure are represented by the Nodes, connected using links or pointers.
- Each node contains two fields, the information field consists of the actual data, and the pointer field consists of the address of the subsequent nodes in the list.
- The pointer of the last node of the linked list consists of a null pointer, as it points to nothing. Unlike the Arrays, the user can dynamically adjust the size of a Linked List as per the requirements.



Linked Lists can be classified into different types:

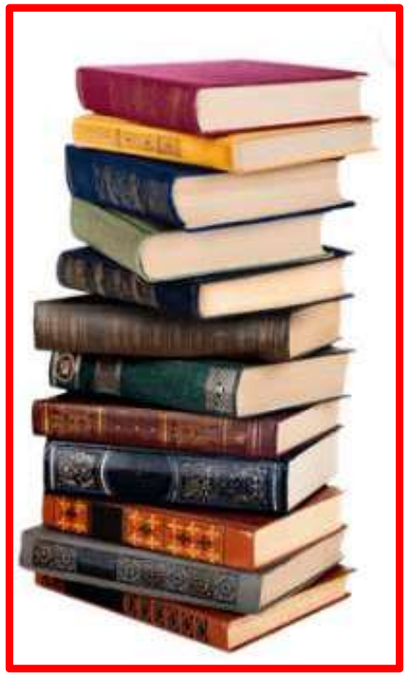
1. **Singly Linked List:** A Singly Linked List is the most common type of Linked List. Each node has data and a pointer field containing an address to the next node.
2. **Doubly Linked List:** A Doubly Linked List consists of an information field and two pointer fields. The information field contains the data. The first pointer field contains an address of the previous node, whereas another pointer field contains a reference to the next node. Thus, we can go in both directions (backward as well as forward).
3. **Circular Linked List:** The Circular Linked List is similar to the Singly Linked List. The only key difference is that the last node contains the address of the first node, forming a circular loop in the Circular Linked List.

Some Applications of Linked Lists:

1. The Linked Lists help us implement stacks, queues, binary trees, and graphs of predefined size.
2. We can also implement Operating System's function for dynamic memory management.
3. Linked Lists also allow polynomial implementation for mathematical operations.
4. We can use Circular Linked List to implement Operating Systems or application functions that Round Robin execution of tasks.
5. Circular Linked List is also helpful in a Slide Show where a user requires to go back to the first slide after the last slide is presented.
6. Doubly Linked List is utilized to implement forward and backward buttons in a browser to move forward and backward in the opened pages of a website.

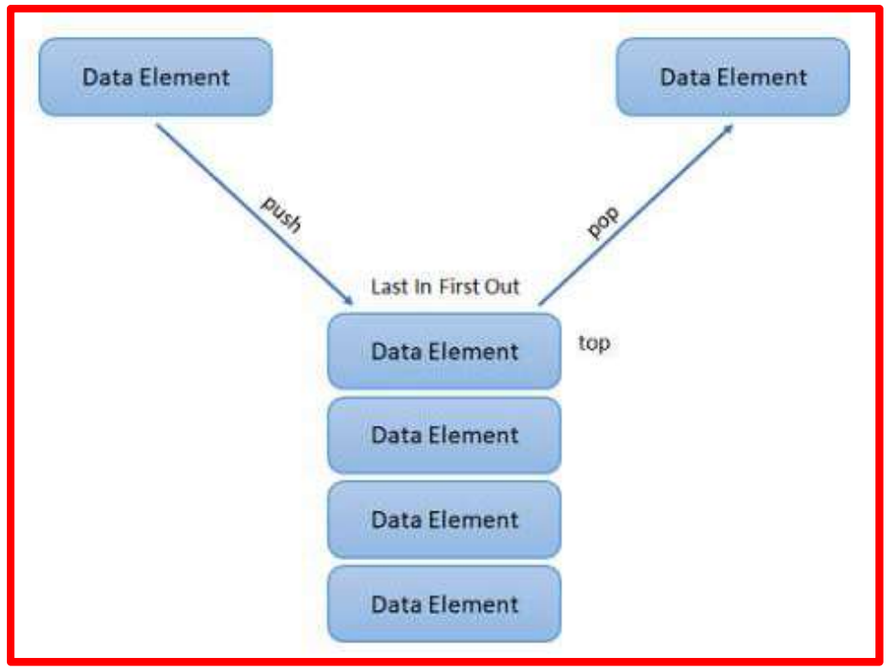
3. Stacks

- A **Stack** is a Linear Data Structure that follows the **LIFO** (Last In, First Out) principle that allows operations like insertion and deletion from one end of the Stack, i.e., Top.
- Stacks can be implemented with the help of contiguous memory, an Array, and non-contiguous memory, a Linked List. Real-life examples of Stacks are piles of books, a deck of cards, piles of money, and many more.



The primary operations in the Stack are as follows:

- **Push:** Operation to insert a new element in the Stack is termed as Push Operation.
- **Pop:** Operation to remove or delete elements from the Stack is termed as Pop Operation.



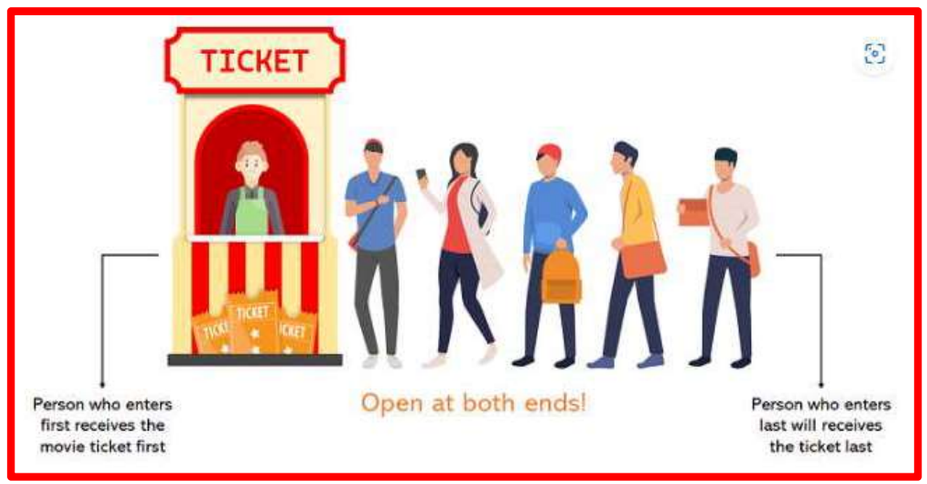
Some Applications of Stacks:

1. The Stack is used as a Temporary Storage Structure for recursive operations.
2. Stack is also utilized as Auxiliary Storage Structure for function calls, nested operations, and deferred/postponed functions.
3. We can manage function calls using Stacks.
4. Stacks are also utilized to evaluate the arithmetic expressions in different programming languages.
5. Stacks are also helpful in converting infix expressions to postfix expressions.
6. Stacks allow us to check the expression's syntax in the programming environment.
7. We can match parenthesis using Stacks.

8. Stacks can be used to reverse a String.
9. Stacks are helpful in solving problems based on backtracking.
10. We can use Stacks in depth-first search in graph and tree traversal.
11. Stacks are also used in Operating System functions.
12. Stacks are also used in UNDO and REDO functions in an edit.

4. Queues

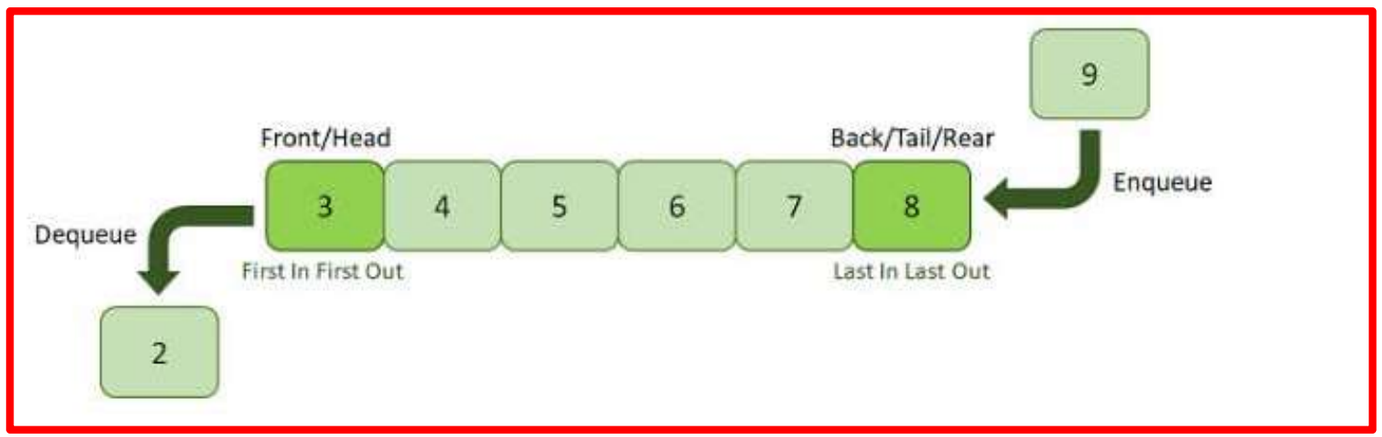
- A **Queue** is a linear data structure similar to a Stack with some limitations on the insertion and deletion of the elements. The insertion of an element in a Queue is done at one end, and the removal is done at another or opposite end.
- Thus, we can conclude that the Queue data structure follows FIFO (First In, First Out) principle to manipulate the data elements. Implementation of Queues can be done using Arrays, Linked Lists, or Stacks. Some real-life examples of Queues are a line at the ticket counter, an escalator, a car wash, and many more.



The following are the primary operations of the Queue:

A. **Enqueue:** The insertion or Addition of some data elements to the Queue is called Enqueue. The element insertion is always done with the help of the rear pointer.

B. **Dequeue:** Deleting or removing data elements from the Queue is termed Dequeue. The deletion of the element is always done with the help of the front pointer.



Some Applications of Queues:

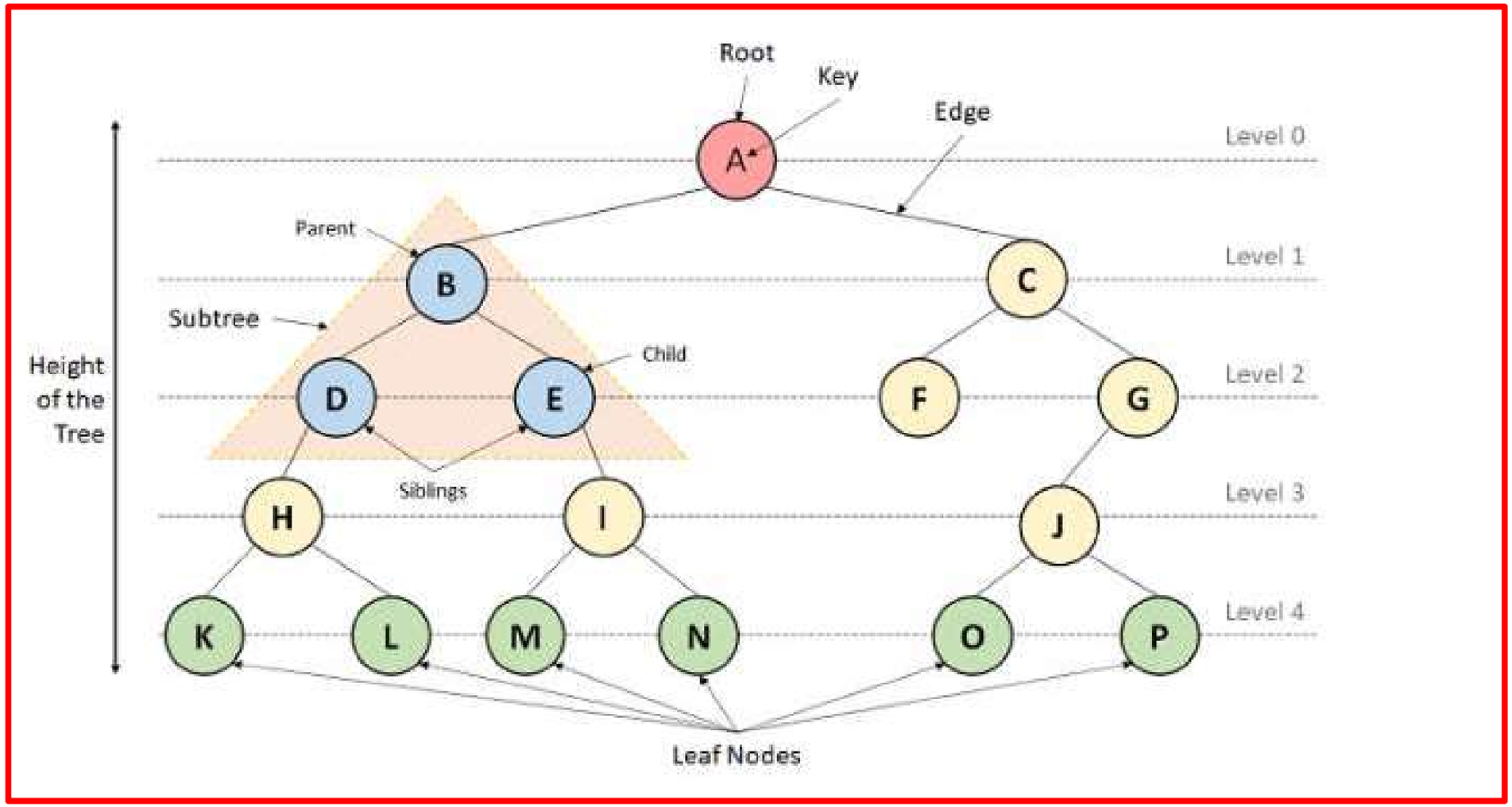
1. Queues are generally used in the breadth search operation in Graphs.
2. Queues are also used in Job Scheduler Operations of Operating Systems, like a keyboard buffer queue to store the keys pressed by users and a print buffer queue to store the documents printed by the printer.
3. Queues are responsible for CPU scheduling, Job scheduling, and Disk Scheduling.
4. Priority Queues are utilized in file-downloading operations in a browser.
5. Queues are also used to transfer data between peripheral devices and the CPU.
6. Queues are also responsible for handling interrupts generated by the User Applications for the CPU.

Non Linear Data Structures

- Non-Linear Data Structures are data structures where the data elements are not arranged in sequential order. Here, the insertion and removal of data are not feasible in a linear manner. There exists a hierarchical relationship between the individual data items.
  - The best example is trees and graphs. In this case, the elements are arranged in a random manner.

1. Trees

- A Tree is a Non-Linear Data Structure and a hierarchy containing a collection of nodes such that each node of the tree stores a value and a list of references to other nodes (the "children").
- The Tree data structure is a specialized method to arrange and collect data in the computer to be utilized more effectively. It contains a central node, structural nodes, and sub-nodes connected via edges. We can also say that the tree data structure consists of roots, branches, and leaves connected.


Root
Key
Edge
Level 0
A
Parent
B
C
Level 1
Subtree
Child
Height of the Tree
D
E
F
G
Level 2
Siblings
H
I
J
Level 3
K
L
M
N
O
P
Level 4
Leaf Nodes

Trees can be classified into different types:

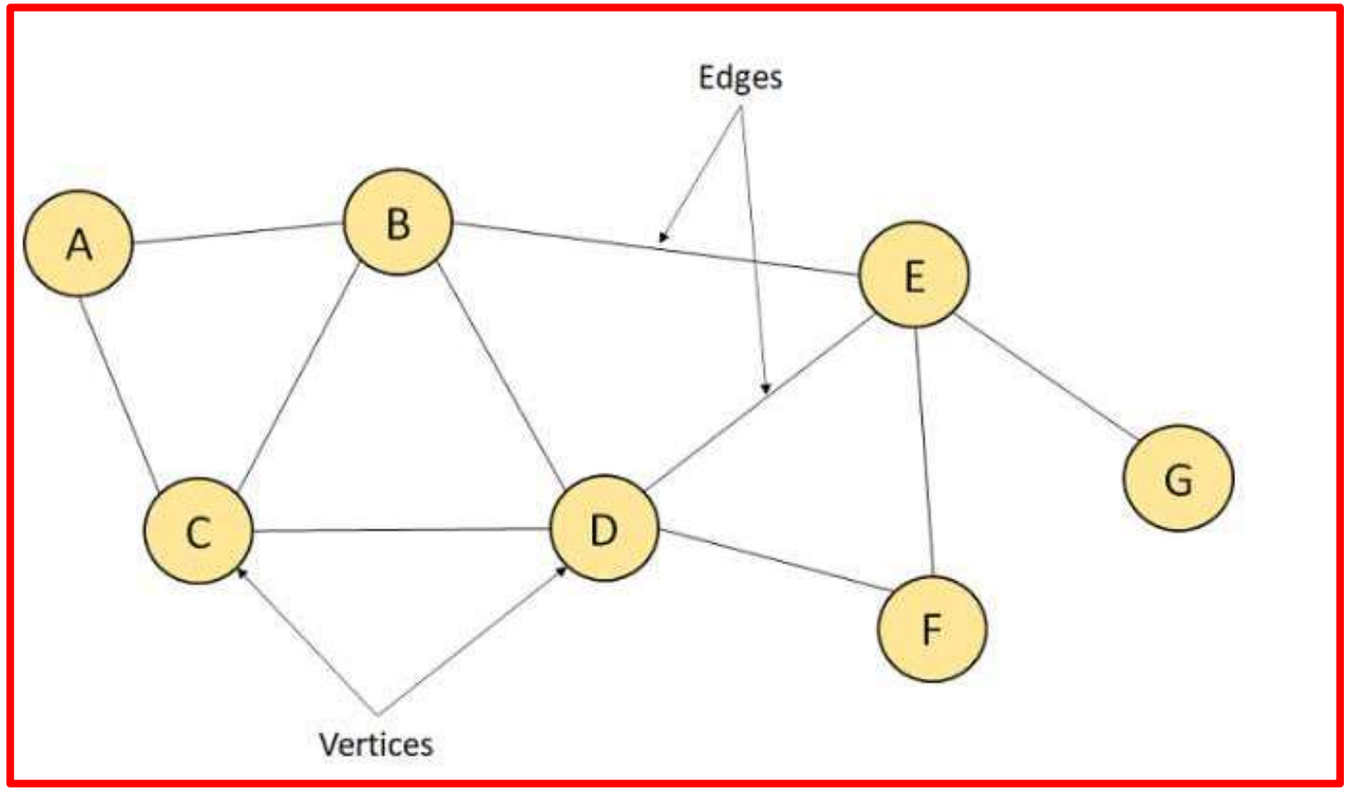
1. **Binary Tree:** A Tree data structure where each parent node can have at most two children is termed a Binary Tree.
2. **Binary Search Tree:** A Binary Search Tree is a Tree data structure where we can easily maintain a sorted list of numbers.
3. **AVL Tree:** An AVL Tree is a self-balancing Binary Search Tree where each node maintains extra information known as a Balance Factor whose value is either -1, 0, or +1.
4. **B-Tree:** A B-Tree is a special type of self-balancing Binary Search Tree where each node consists of multiple keys and can have more than two children.

Some Applications of Trees:

1. Trees implement hierarchical structures in computer systems like directories and file systems.
2. Trees are also used to implement the navigation structure of a website.
3. We can generate code like Huffman's code using Trees.
4. Trees are also helpful in decision-making in Gaming applications.
5. Trees are responsible for implementing priority queues for priority-based OS scheduling functions.
6. Trees are also responsible for parsing expressions and statements in the compilers of different programming languages.
7. We can use Trees to store data keys for indexing for Database Management System (DBMS).
8. Spanning Trees allows us to route decisions in Computer and Communications Networks.
9. Trees are also used in the path-finding algorithm implemented in Artificial Intelligence (AI), Robotics, and Video Games Applications.

2. Graphs

- A Graph is another example of a Non-Linear Data Structure comprising a finite number of nodes or vertices and the edges connecting them. The Graphs are utilized to address problems of the real world in which it denotes the problem area as a network such as social networks, circuit networks, and telephone networks. For instance, the nodes or vertices of a Graph can represent a single user in a telephone network, while the edges represent the link between them via telephone.
- The Graph data structure, G is considered a mathematical structure comprised of a set of vertices, V and a set of edges, E



- In the figure represents a Graph having seven vertices A, B, C, D, E, F, G, and ten edges [A, B], [A, C], [B, C], [B, D], [B, E], [C, D], [D, E], [D, F], [E, F], and [E, G].

Depending upon the position of the vertices and edges, the Graphs can be classified into different types:

1. **Null Graph:** A Graph with an empty set of edges is termed a Null Graph.
2. **Trivial Graph:** A Graph having only one vertex is termed a Trivial Graph.
3. **Simple Graph:** A Graph with neither self-loops nor multiple edges is known as a Simple Graph.
4. **Multi Graph:** A Graph is said to be Multi if it consists of multiple edges but no self-loops.
5. **Pseudo Graph:** A Graph with self-loops and multiple edges is termed a Pseudo Graph.
6. **Non-Directed Graph:** A Graph consisting of non-directed edges is known as a Non-Directed Graph.
7. …

Some Applications of Graphs:

1. Graphs help us represent routes and networks in transportation, travel, and communication applications.
2. Graphs are used to display routes in GPS.
3. Graphs also help us represent the interconnections in social networks and other network-based applications.
4. Graphs are utilized in mapping applications.
5. Graphs are responsible for the representation of user preference in e-commerce applications.
6. Graphs are also used in Utility networks in order to identify the problems posed to local or municipal corporations.
7. Graphs also help to manage the utilization and availability of resources in an organization.
8. Graphs are also used to make document link maps of the websites in order to display the connectivity between the pages through hyperlinks.
9. Graphs are also used in robotic motions and neural networks.

Major Operations

1. **Searching:** We can search for any element in a data structure.
2. **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
3. **Insertion:** We can also insert the new element in a data structure.
4. **Updation:** We can also update the element, i.e., we can replace the element with another element.
5. **Deletion:** We can also perform the delete operation to remove the element from the data structure.

6. **Traversal:** Traversing a data structure means accessing each data element exactly once so it can be administered. For example, traversing is required while printing the names of all the employees in a department.
7. **Merge:** Merge means to combine data elements of two sorted lists in order to form a single list of sorted data elements.
8. **Create:** Create is an operation used to reserve memory for the data elements of the program. We can perform this operation using a declaration statement.
9. **Selection:** Selection means selecting a particular data from the available data. We can select any particular data by specifying conditions inside the loop.
10. **Splitting:** The Splitting operation allows us to divide data into various subparts decreasing the overall process completion time.

Which Data Structure?

- A data structure is a way of organizing the data so that it can be used efficiently. Here, we have used the word efficiently, which in terms of both the space and time. For example, a stack is an ADT (Abstract data type) which uses either arrays or linked list data structure for the implementation. Therefore, we conclude that we require some data structure to implement a particular ADT.
- An ADT tells **what** is to be done and data structure tells **how** it is to be done. In other words, we can say that ADT gives us the blueprint while data structure provides the implementation part. Now the question arises: how can one get to know which data structure to be used for a particular ADT?.

- As the different data structures can be implemented in a particular ADT, but the different implementations are compared for time and space. For example, the Stack ADT can be implemented by both Arrays and linked list. Suppose the array is providing time efficiency while the linked list is providing space efficiency, so the one which is the best suited for the current user's requirements will be selected.

Advantages of Data structures

The following are the advantages of a data structure:

- **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
- **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.
- **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

Some Applications of Data Structures

1. Data Structures help in the organization of data in a computer's memory.
2. Data Structures also help in representing the information in databases.
3. Data Structures allows the implementation of algorithms to search through data (For example, search engine).
4. We can use the Data Structures to implement the algorithms to manipulate data (For example, word processors).
5. We can also implement the algorithms to analyse data using Data Structures (For example, data miners).
6. Data Structures support algorithms to generate the data (For example, a random number generator).

8. Data Structures also support algorithms to compress and decompress the data (For example, a zip utility).
9. We can also use Data Structures to implement algorithms to encrypt and decrypt the data (For example, a security system).
10. With the help of Data Structures, we can build software that can manage files and directories (For example, a file manager).
11. We can also develop software that can render graphics using Data Structures. (For example, a web browser or 3D rendering software).

Summary

In this lesson, you should have learned that:

- Introduction to Data Structures
- Objects and Collections
- Constructors and Destructors
- Types of Data Structures (Arrays …)