



# Bharathidasan University

Centre for Differently Abled Persons

Tiruchirappalli – 620024.

- ▶ Programme Name : Bachelor of Computer Applications
- ▶ Course Code : 20UCA6CC10
- ▶ Course Title : Data Structures
- ▶ Unit : Unit I
- ▶ Compiled by : Dr. M. Prabavathy  
Associate Professor  
Ms. Patric Matharasi  
Guest Faculty

# UNIT I

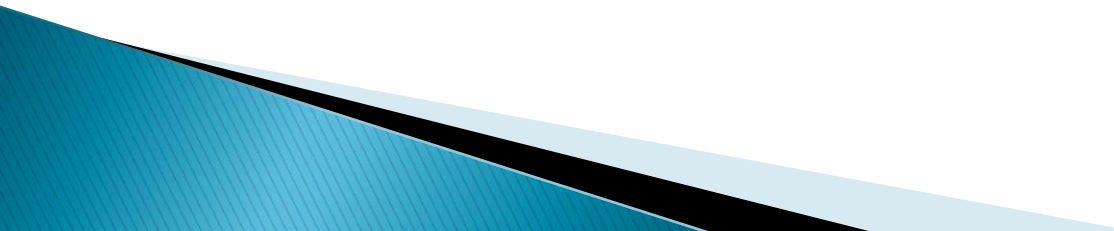
# Data

- ▶ 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.

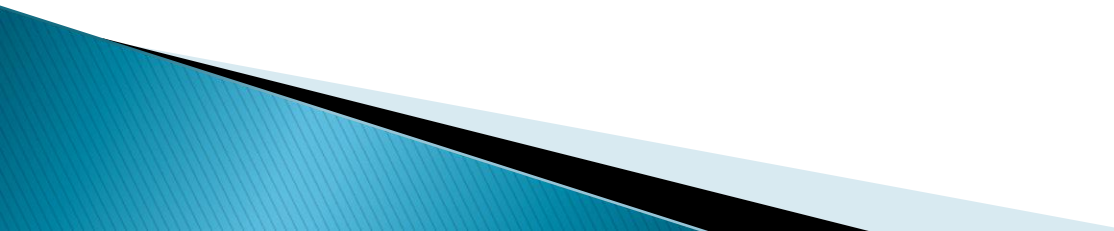
# Data Structure

- ▶ Data Structure is a branch of Computer Science.
- ▶ Data Structure is a particular way of storing and organizing data in the memory of the computer.
- ▶ These data can easily be retrieved and efficiently utilized in the future when required.
- ▶ The data can be managed in various ways, like the logical or mathematical model for a specific organization of data is known as a data structure.

# Need for Data Structures

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


# Objectives of Data Structures

1. **Correctness:** Data Structures are designed to operate correctly for all kinds of inputs based on the domain of interest.
  2. **Efficiency:** It should process the data quickly without utilizing many computer resources like memory space.
- 

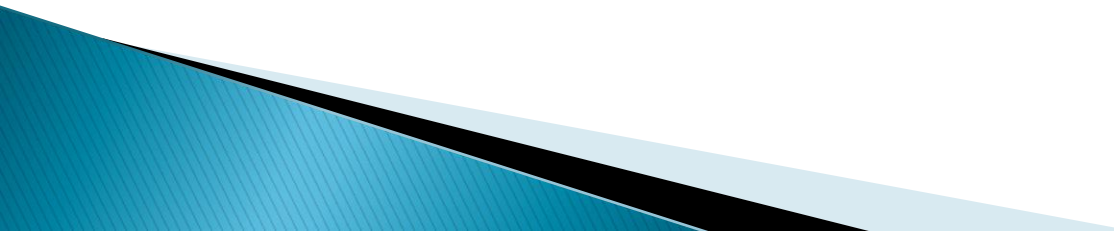
# Features of Data Structures

1. **Robustness:** Generally, all computer programmers aim to produce software that yields correct output for every possible input, along with efficient execution on all hardware platforms.

2. **Adaptability:** Building software applications like Web Browsers, Word Processors, and Internet Search Engine include huge software systems that require correct and efficient working or execution for many years.

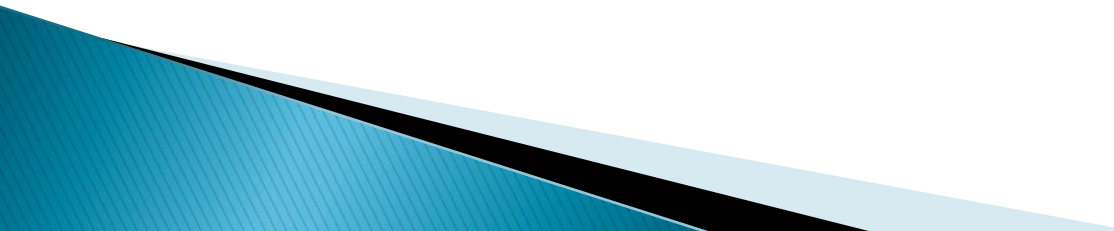


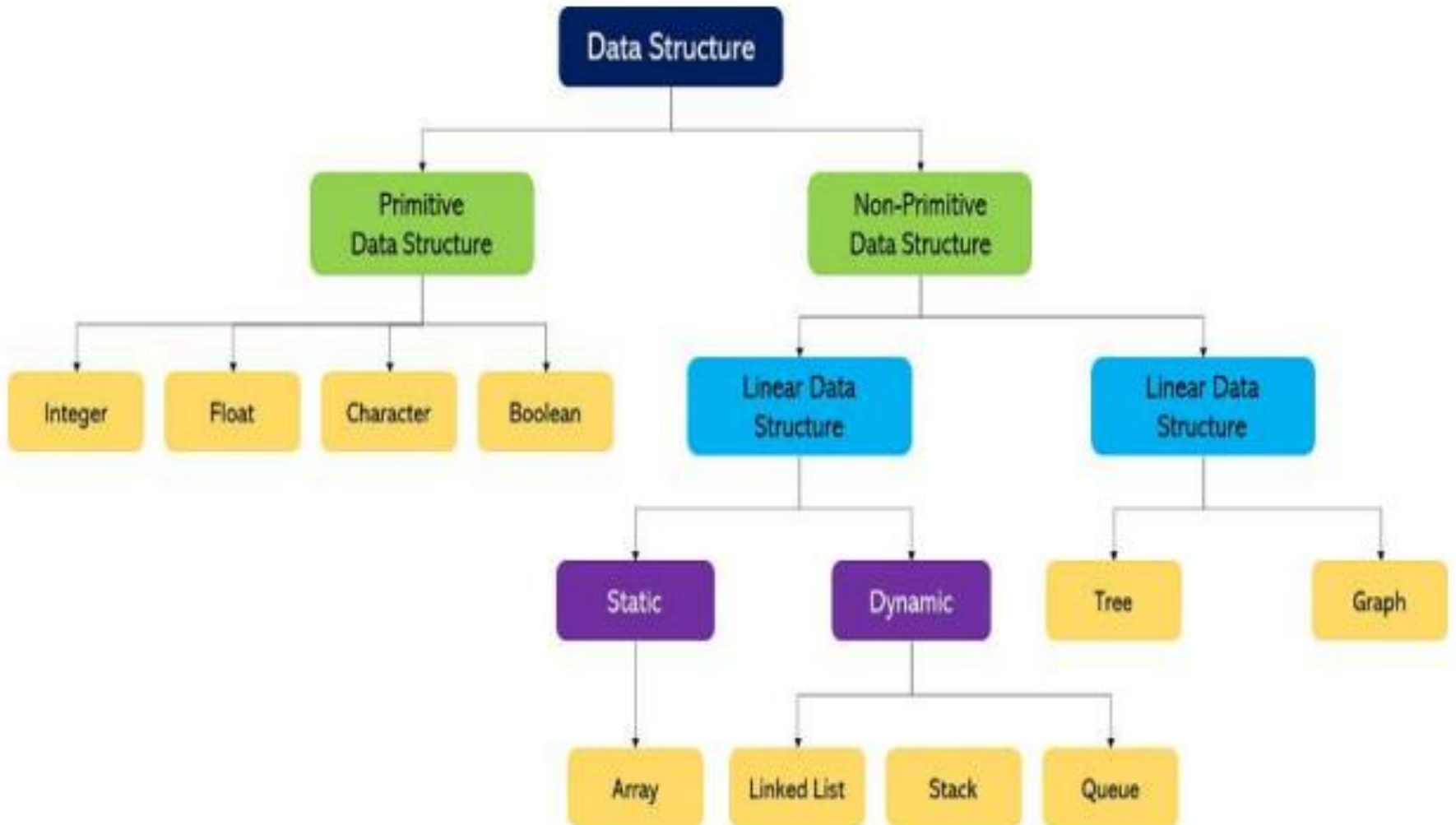
**3. Reusability:** However, if the software is developed in a reusable and adaptable way, then it can be applied in most future applications. Thus, by executing quality data structures, it is possible to build reusable software, which appears to be cost-effective and timesaving.






# Classification of Data Structures

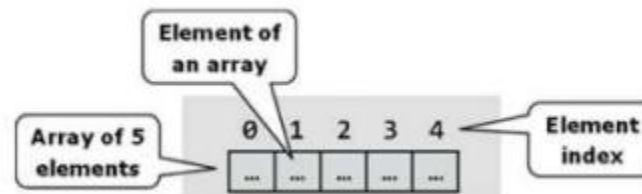
- ▶ We can classify Data Structures into two categories:
    1. Primitive Data Structure
    2. Non-Primitive Data Structure
- 



# Arrays

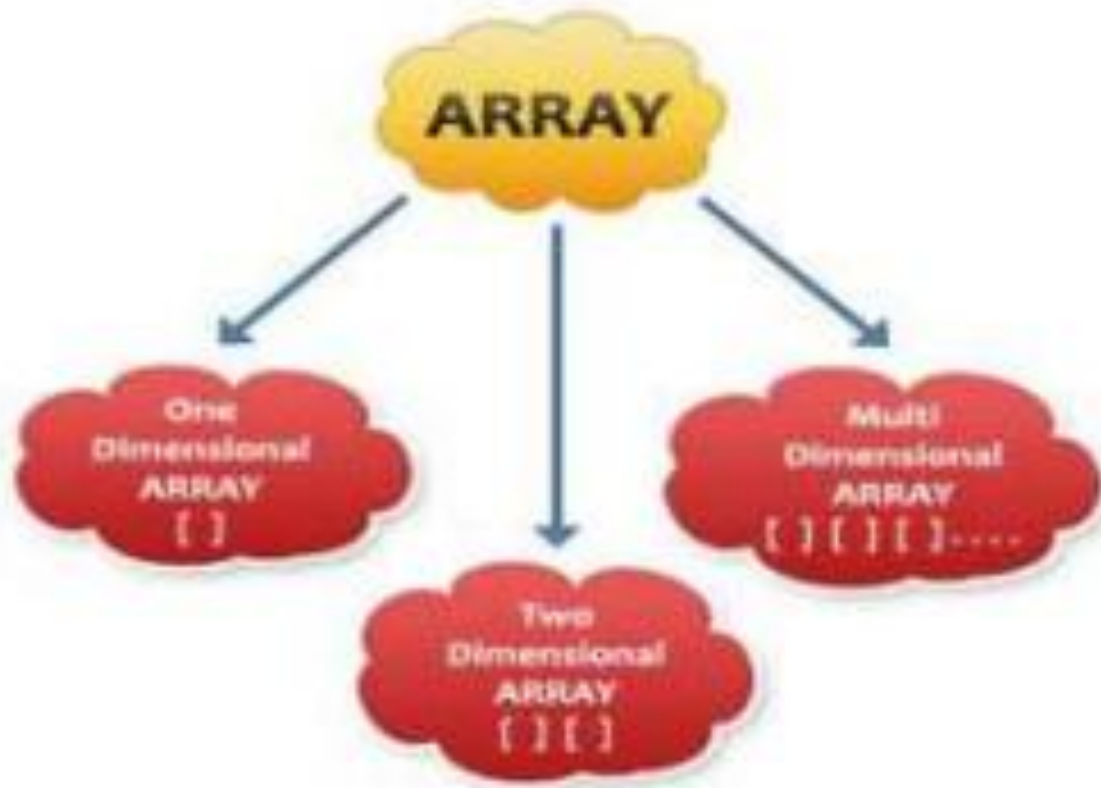
- ▶ A set of finite number of homogeneous elements or same data items
  - ▶ An array can contain one type of data only
  - ▶ Declaration of array is : `int arr[10]`
  - ▶ The number specified inside the square brackets [ ] is the number of elements an array can be store.
  - ▶ The elements of array can be stored in the consecutive (continues) memory location.
- 

This length is given by the following equation:



# Types of Array

- ▶ Array is classified into 3 types



# One Dimension Array

- ▶ An array with only one row or column is called one dimensional array.
- ▶ Syntax: datatype  
array\_name[Size];
- ▶ Example: int arr[3];

Index	1	2	3	4	5	6
Value	15	17	25	90	110	221

# Two Dimension Array

- ▶ In two dimensional arrays the array is divided into rows and columns.
- ▶ Syntax: datatype  
array\_name[row\_size][col\_size];
- ▶ Example: int arr[3][3];

Index	1	2	3
1	10	15	7
2	9	25	30
3	39	2	84

# Multi Dimension Array

- ▶ Array with Multiple subscripts
- ▶ Multidimensional arrays are often known as array of the arrays
- ▶ Syntax: datatype array\_name[size 1][size 2][size 3]-----[size N];

	Index	1	2	3
	Index	1	2	3
Index	1	2	3	31
1	10	15	7	33
2	9	25	30	90
3	39	2	84	



# Basic operations on Arrays

## ❖ Traverse

Processing each element in the array.

## ❖ Search

Finding the location of an element with a given value.

## ❖ Insertion

Adding a new element to an array.

## ❖ Deletion

Removing an element from an array.

## ❖ Sorting

Organizing the elements in some order.

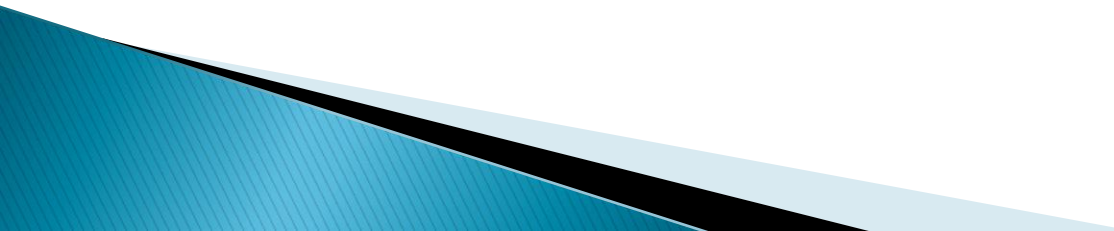
## ❖ Merging

Combining two arrays into a single array.

# Advantages of an Array

- Collection of similar types of data.
- 2 dimensional array is used to represent a matrix .
- It is used to other data structure like linked lists , stacks , queue , etc

# Disadvantages of an Array

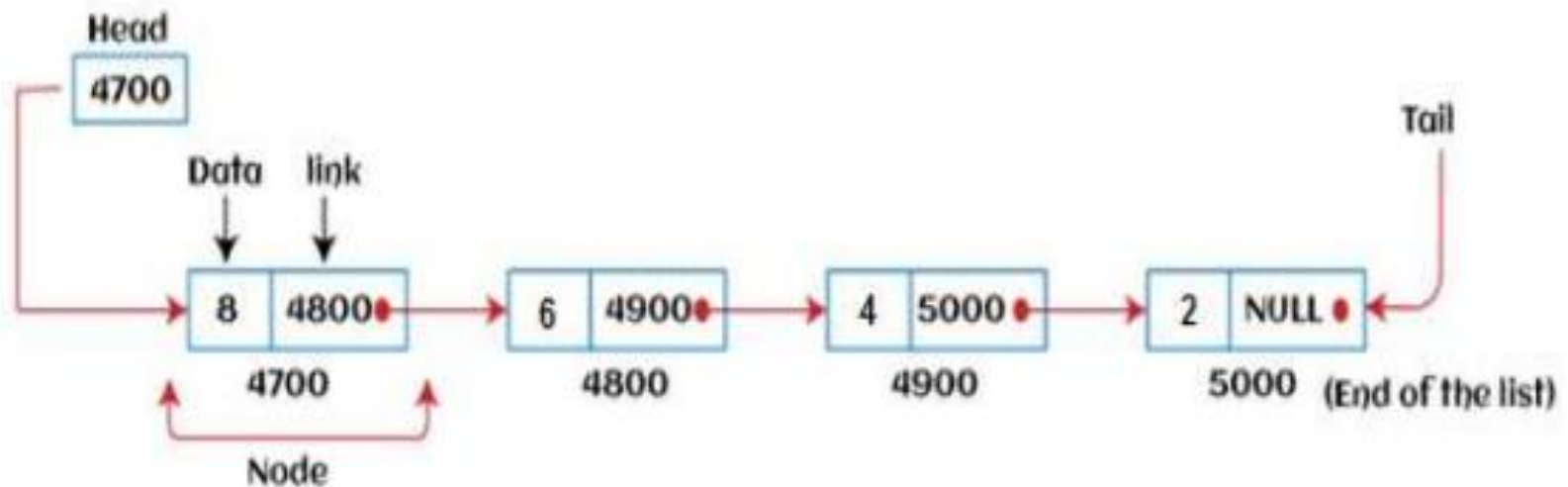
- The time complexity increase in insertion and deletion operation.
  - Wastage of memory because arrays are fixed in size.
- 

# Linked List

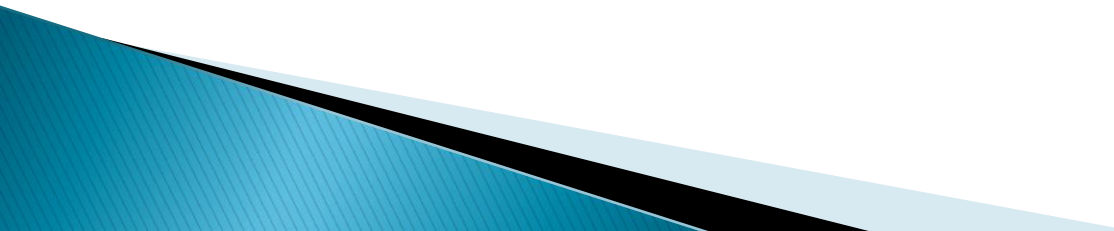
- ❑ Linked list is a linear data structure that includes a series of connected nodes.
- ❑ Linked list can be defined as the nodes that are randomly stored in the memory.
- ❑ A node in the linked list contains two parts, i.e., first is the data part and second is the address part. o The last node of the list contains a pointer to the null.
- ❑ After array, linked list is the second most used data structure.
- ❑ In a linked list, every link contains a connection to another link.

# Representation of a Linked list

- ▶ Linked list can be represented as the connection of nodes in which each node points to the next node of the list.
- ▶ The representation of the linked list is shown below

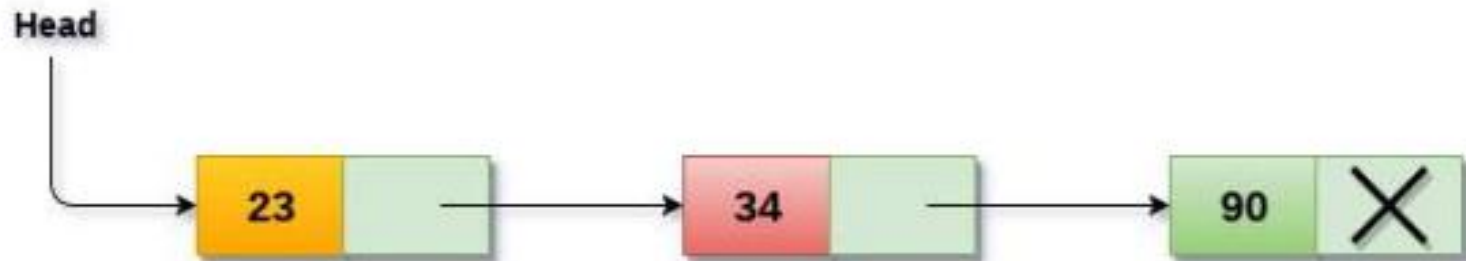


# Types of Linked list

- ▶ **Singly-linked list**
  - ▶ **Doubly linked list**
  - ▶ **Circular singly linked list**
  - ▶ **Circular doubly linked list**
- 

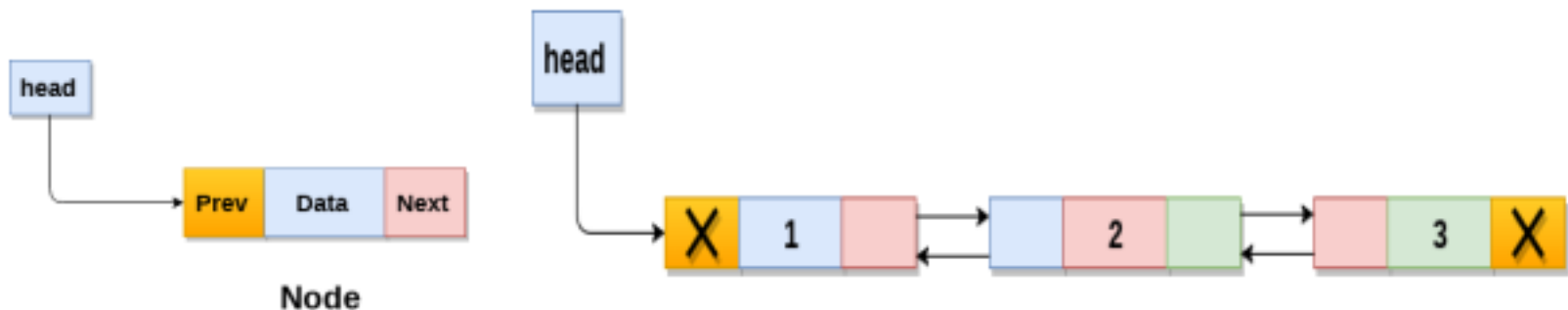
# Singly-linked list

- ▶ Singly linked list can be defined as the collection of an ordered set of elements.
- ▶ A node in the singly linked list consists of two parts: data part and link part.
- ▶ Data part of the node stores actual information that is to be represented by the node, while the link part of the node stores the address of its immediate successor.



# Doubly linked list

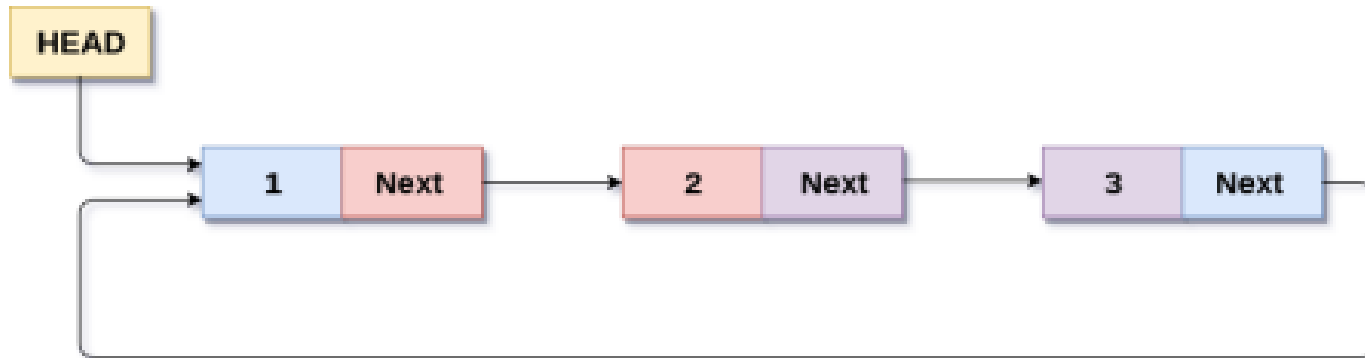
- ▶ Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.
- ▶ Therefore, in a doubly-linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer), and pointer to the previous node (previous pointer).





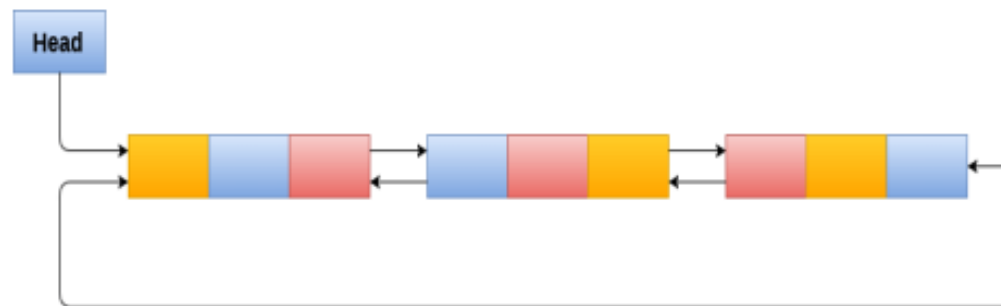
# Circular singly linked list

- ▶ In a circular singly linked list, the last node of the list contains a pointer to the first node of the list.
- ▶ We can have circular singly linked list as well as circular doubly linked list.



# Circular doubly linked list

- ▶ Circular doubly linked list is a more complex type of data structure in which a node contains pointers to its previous node as well as the next node.
- ▶ Circular doubly linked list doesn't contain NULL in any of the nodes.
- ▶ The last node of the list contains the address of the first node of the list.
- ▶ The first node of the list also contains the address of the last node in its previous pointer.



# Benefits of Linked list

- ▶ **Dynamic data structure** – The size of the linked list may vary according to the requirements. Linked list does not have a fixed size.
- ▶ **Insertion and deletion** – Insertion, and deletion in linked list is easier. The elements in the linked list are stored at a random location.
- ▶ **Memory efficient** – The size of a linked list can grow or shrink according to the requirements, so memory consumption in linked list is efficient.
- ▶ **Implementation** – We can implement both stacks and queues using linked list.

# Limitations of Linked list

- ▶ **Memory usage** – In linked list, node occupies more memory than array. Each node of the linked list occupies two types of variables, i.e., one is a simple variable, and another one is the pointer variable.
- ▶ **Traversal** – Traversal is not easy in the linked list. The time required to access a particular node is large.
- ▶ **Reverse traversing** – Backtracking or reverse traversing is difficult in a linked list. In a doubly-linked list, it is easier but requires more memory to store the back pointer

**THANK YOU**

