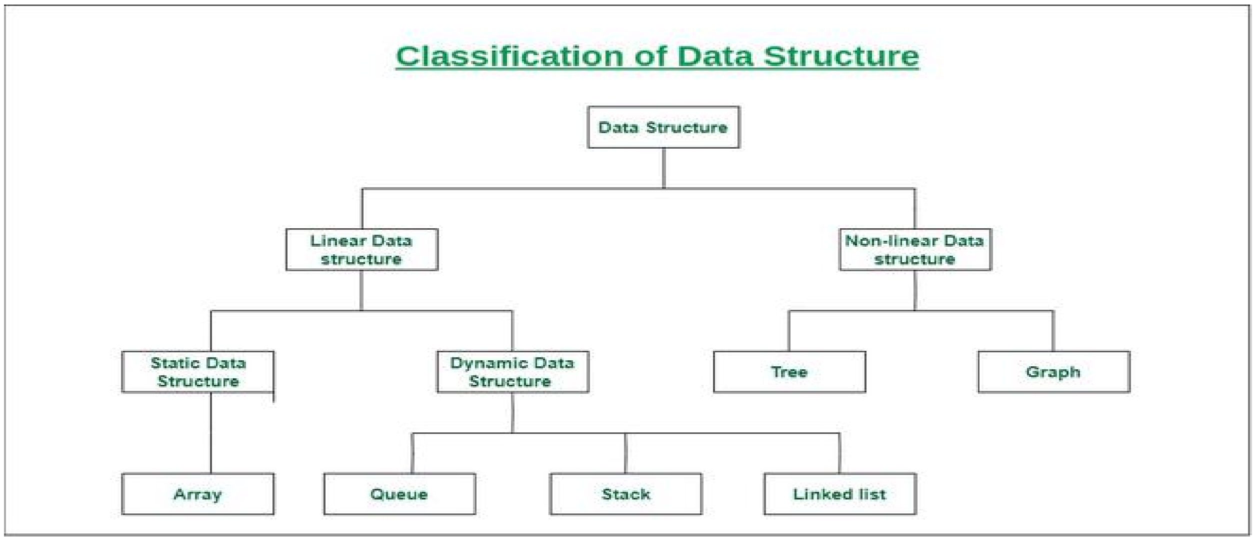
DS-UNIT 1

1. What is data structure?

Ans-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.



2. Why do we need data structure?

Ans-Data Structures are necessary for designing efficient algorithms. It provides reusability and abstraction. Using appropriate data structures can help programmers save a good amount of time while performing operations such as storage, retrieval, or processing of data.

3. List and explain any two data structures.

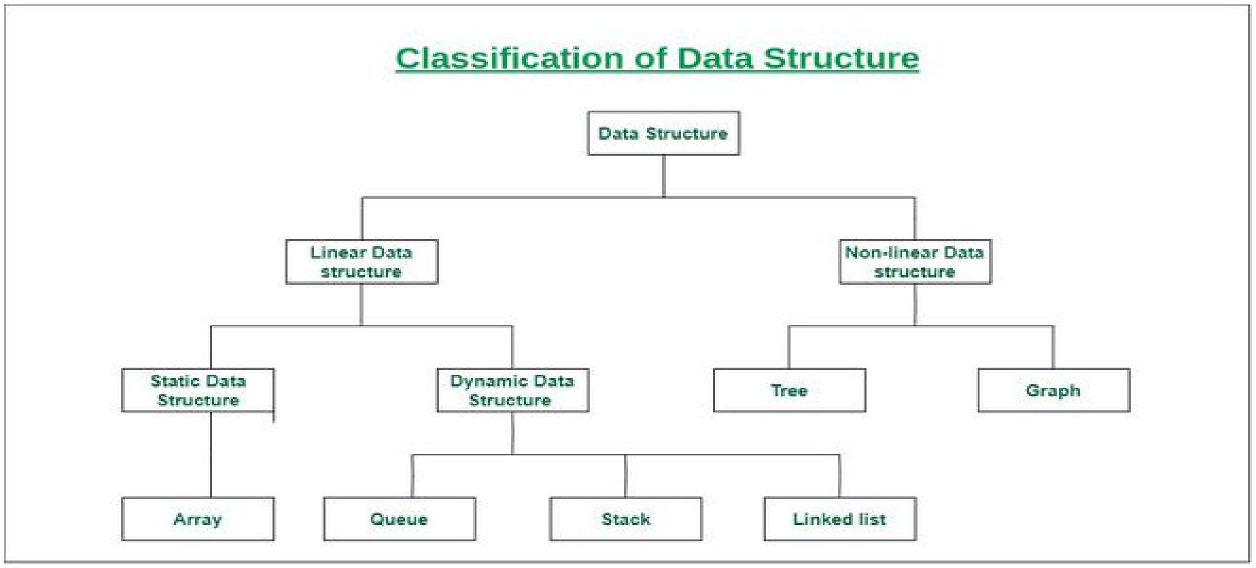
Ans-Linear data structure: Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.

- Static data structure: Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
- Dynamic data structure: In 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.

Non-linear data structure: Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can’t traverse all the elements in a single run only.

4. How are data structures classified? Explain it.

Ans-



Linear data structure: Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.

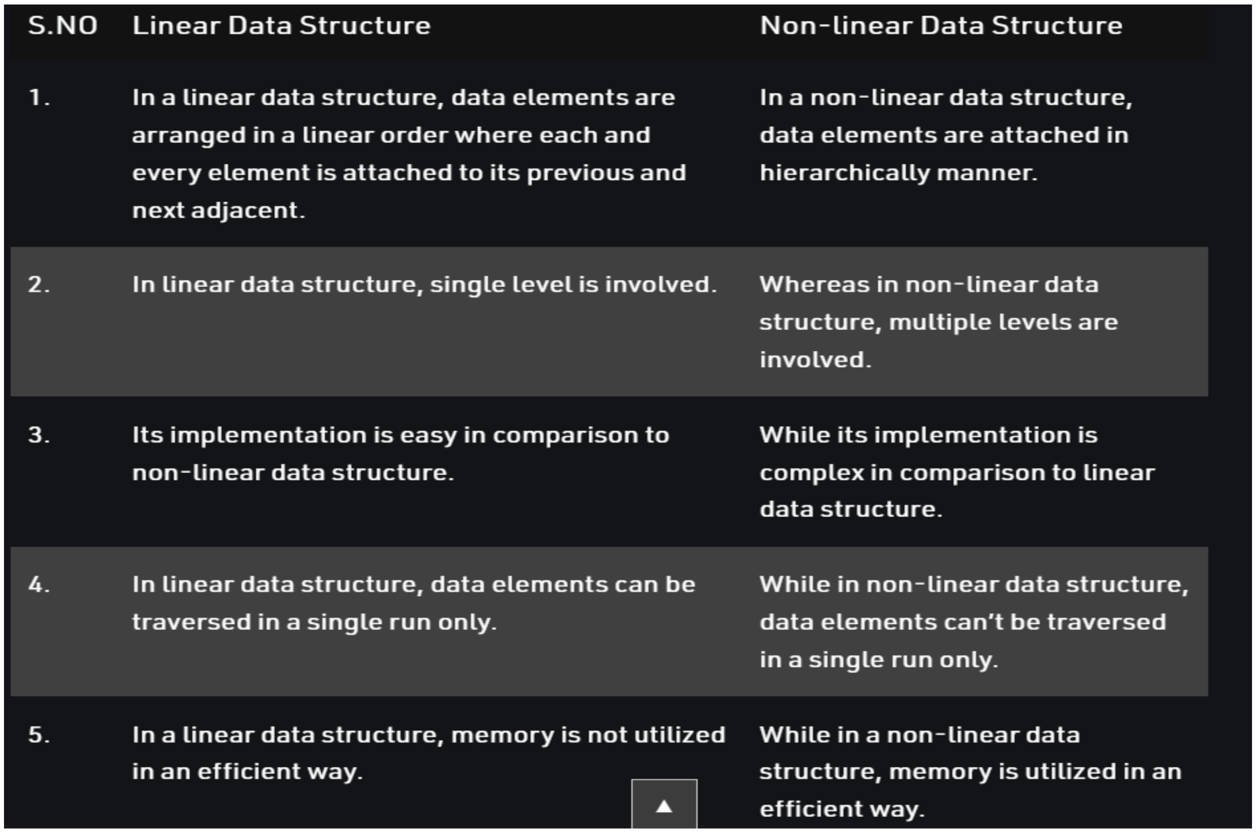
Examples of linear data structures are array, stack, queue, linked list, etc.

Non-linear data structure: Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can't traverse all the elements in a single run only.

Examples of non-linear data structures are trees and graphs.

5. Differentiate between linear and non-linear data structures.

Ans-



S.NO	Linear Data Structure	Non-linear Data Structure
1.	In a linear data structure, data elements are arranged in a linear order where each and every element is attached to its previous and next adjacent.	In a non-linear data structure, data elements are attached in hierarchically manner.
2.	In linear data structure, single level is involved.	Whereas in non-linear data structure, multiple levels are involved.
3.	Its implementation is easy in comparison to non-linear data structure.	While its implementation is complex in comparison to linear data structure.
4.	In linear data structure, data elements can be traversed in a single run only.	While in non-linear data structure, data elements can't be traversed in a single run only.
5.	In a linear data structure, memory is not utilized in an efficient way.	While in a non-linear data structure, memory is utilized in an efficient way.

6. Define ADT? Explain features of ADT.

Ans-An ADT is a mathematical model of a data structure that specifies the type of data stored, the operations supported on them, and the types of parameters of the operations. An ADT specifies what each operation does, but not how it does it. Typically, an ADT can be implemented using one of many different data structures.

Features of ADT:

- Abstraction: The user does not need to know the implementation of the data structure.
- Better Conceptualization: ADT gives us a better conceptualization of the real world.
- Robust: The program is robust and has the ability to catch errors.

7. List and explain types of linked list.

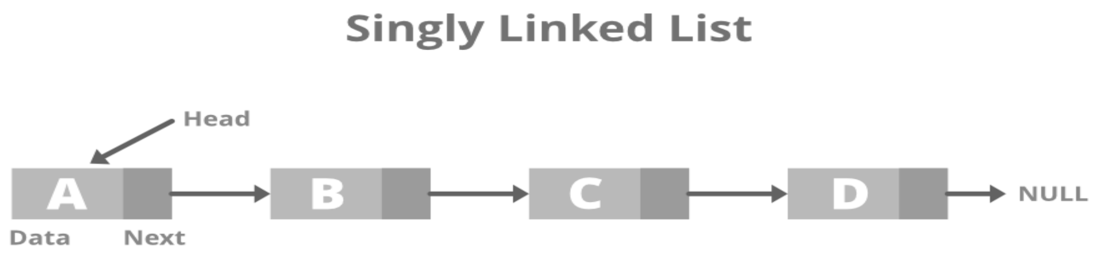
Ans-A linked list is a sequence of data structures, which are connected together via links.

Types of Linked List

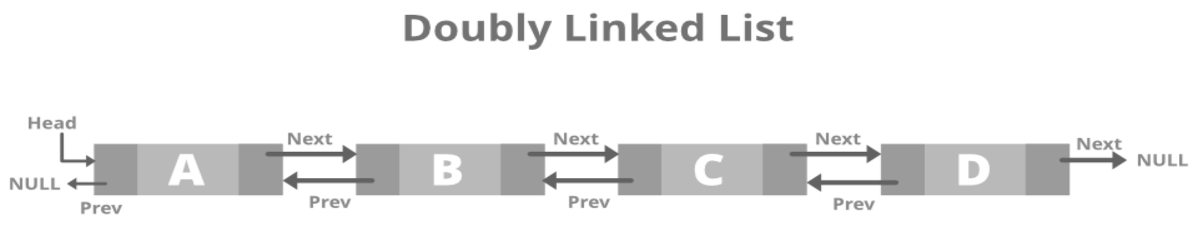
Following are the various types of linked list.

- **Singly Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.
- **Doubly Circular Linked List**-The list can be traversed from both directions i.e. from head to tail or from tail to head.

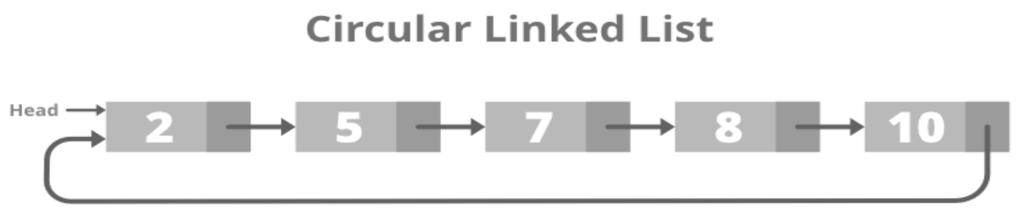
Singly Linked List-It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.



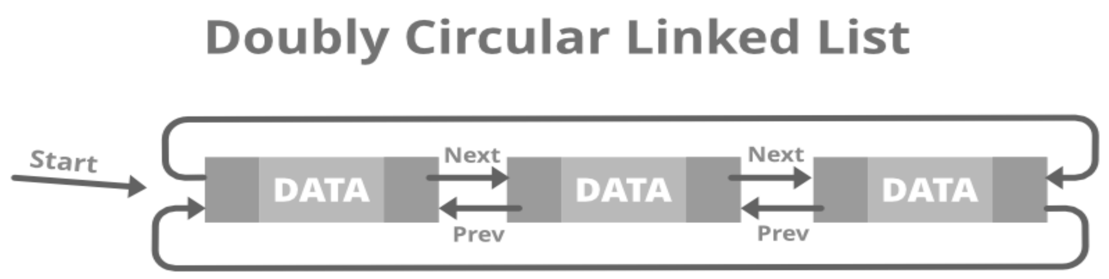
Doubly Linked List-A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence.



Circular Linked List-A circular linked list is that in which the last node contains the pointer to the first node of the list.



Doubly Circular Linked List-A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence.



8. List and explain abstract data types.

Ans-Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.

List ADT

The data is generally stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list.

The List ADT Functions is given below:

- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.
- replace() – Replace an element at any position by another element.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise return false.
- isFull() – Return true if the list is full, otherwise return false.

Stack ADT

In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.

The program allocates memory for the data and address is passed to the stack ADT.

The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.

- push() – Insert an element at one end of the stack called top.
- pop() – Remove and return the element at the top of the stack, if it is not empty.
- peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
- size() – Return the number of elements in the stack.
- isEmpty() – Return true if the stack is empty, otherwise return false.
- isFull() – Return true if the stack is full, otherwise return false.

Queue ADT
The queue abstract data type (ADT) follows the basic design of the stack abstract data type. Each node contains a void pointer to the data and the link pointer to the next element in the queue. The program's responsibility is to allocate memory for storing the data.

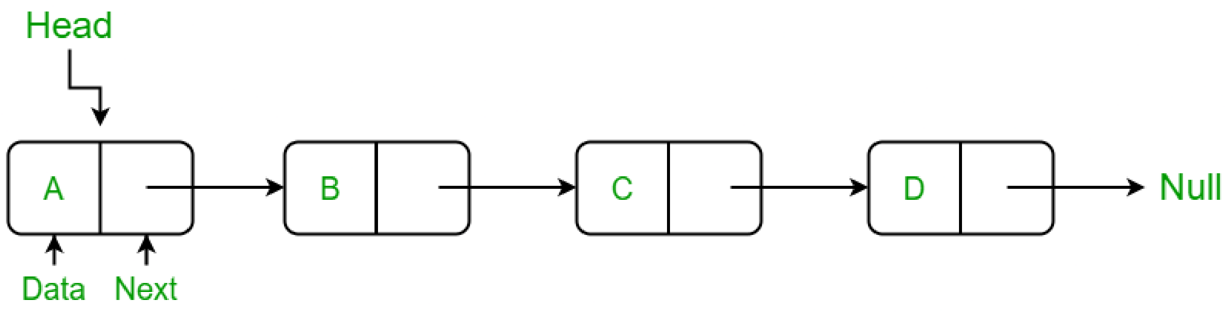
- enqueue() – Insert an element at the end of the queue.
- dequeue() – Remove and return the first element of the queue, if the queue is not empty.
- peek() – Return the element of the queue without removing it, if the queue is not empty.
- size() – Return the number of elements in the queue.
- isEmpty() – Return true if the queue is empty, otherwise return false.
- isFull() – Return true if the queue is full, otherwise return false.

9. How to represent singly linked list? Explain it?

Ans-A linked list is represented by a pointer to the first node of the linked list. The first node is called the head of the linked list. If the linked list is empty, then the value of the head points to NULL.

Each node in a list consists of at least two parts:

- A Data Item (we can store integer, strings, or any type of data).
- Pointer (Or Reference) to the next node (connects one node to another) or An address of another node.



10. Briefly explain working of singly linked list.

Ans-A **singly linked list** is a type of linked list that is *unidirectional*, that is, it can be traversed in only one direction from head to the last node (tail).

Each element in a linked list is called a **node**. A single node contains *data* and a pointer to the *next* node which helps in maintaining the structure of the list.

The first node is called the **head**; it points to the first node of the list and helps us access every other element in the list. The last node, also sometimes called the **tail**, points to *NULL* which helps us in determining when the list ends.

11. Explain how to perform operations on linked list.

Ans-Basic Operations on Linked List

- **Traversal**: To traverse all the nodes one after another.
- **Insertion**: To add a node at the given position.
- **Deletion**: To delete a node.
- **Searching**: To search an element(s) by value.
- **Updating**: To update a node.
- **Sorting:** To arrange nodes in a linked list in a specific order.
- **Merging:** To merge two linked lists into one.

Traverse a Linked List

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

1. Insert at the beginning

Allocate memory for new node

Store data

Change next of new node to point to head

Change head to point to recently created node

2. Insert at the End

Allocate memory for new node

Store data

Traverse to last node

Change next of last node to recently created node

3. Insert at the Middle

Allocate memory and store data for new node

Traverse to node just before the required position of new node

Change next pointers to include new node in between

12. List and explain advantages and disadvantages of linked list.

Ans-Advantages Of Linked List:

Dynamic data structure: A linked list is a dynamic arrangement so it can grow and shrink at runtime by allocating and deallocating memory

No memory wastage: In the Linked list, efficient memory utilization can be achieved since the size of the linked list increase or decrease at run time so there is no memory wastage and there is no need to pre-allocate the memory.

Implementation: Linear data structures like stacks and queues are often easily implemented using a linked list.

Insertion and Deletion Operations: Insertion and deletion operations are quite easier in the linked list.

Disadvantages Of Linked List:

Memory usage: More memory is required in the linked list as compared to an array.

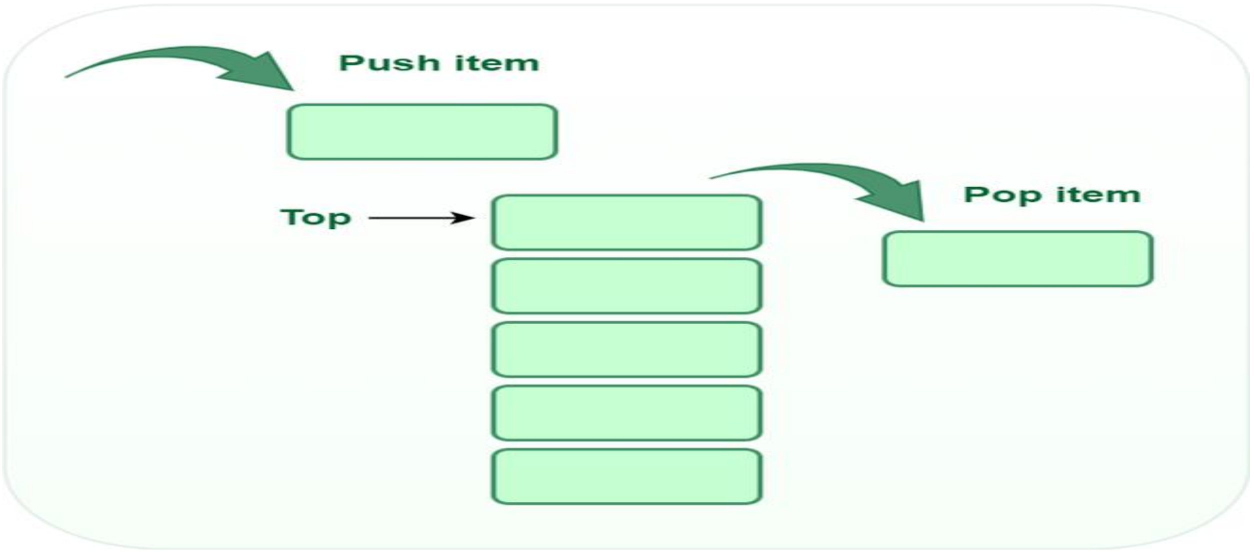
Traversal: In a Linked list traversal is more time-consuming as compared to an array.

Random Access: Random access is not possible in a linked list due to its dynamic memory allocation.

13. How to represent Stack? Explain it?

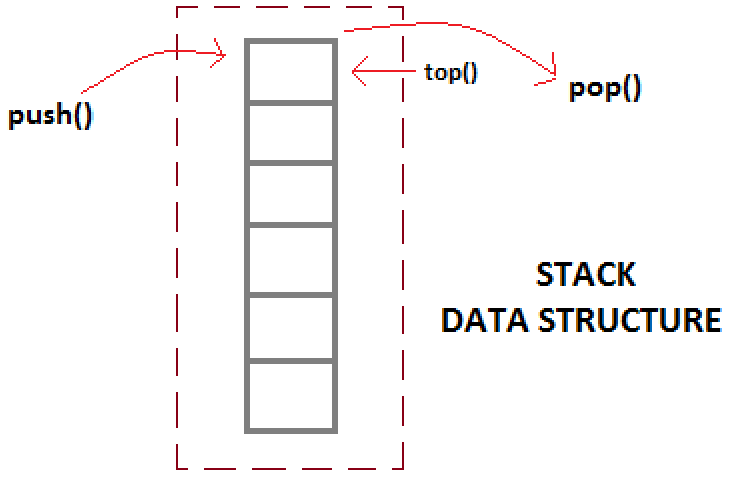
Ans-A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.



14. Briefly explain working of Stack.

Ans-A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end, whereas the Queue has two ends (front and rear). It contains only one pointer top pointer pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack.



15. Explain how to perform operations on Stack.

Ans-It is a linear data structure that follows a particular order in which the operations are performed.

It works on LIFO(Last In First Out) principle.

Push: Adds an item to the stack.

Pop: Removes an item from the stack.

Top: Returns the top element of the stack.

isEmpty: Returns true if the stack is empty, else false.

16. List and explain advantages and disadvantages of Stack.

Ans-Advantages of Stack:

- Stack helps in managing data that follows the LIFO technique.
- Stacks are be used for systematic Memory Management.
- It is used in many virtual machines like JVM.
- Stack cleans up the objects automatically.

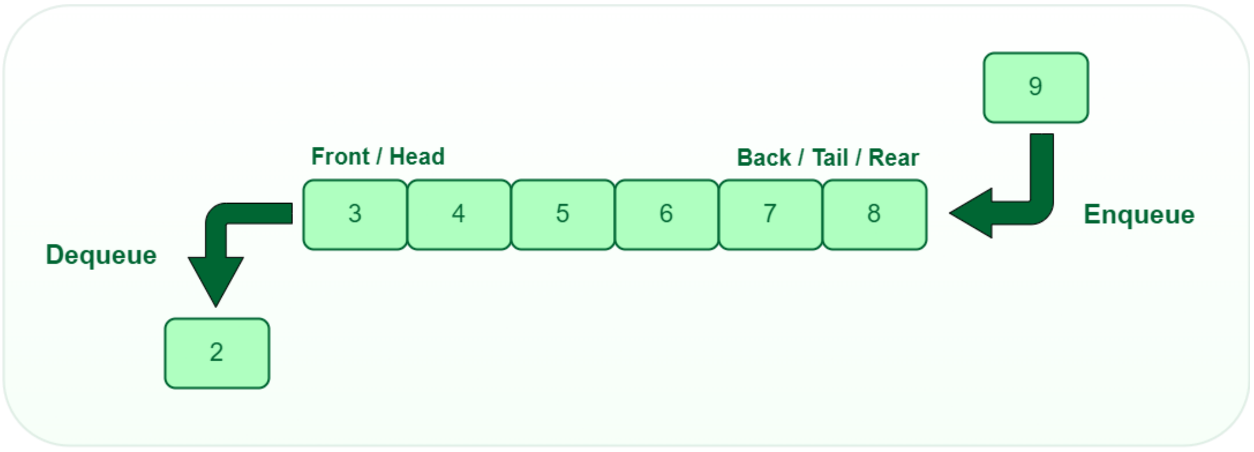
Disadvantages of Stack:

- Stack memory is of limited size.
- The total of size of the stack must be defined before.
- If too many objects are created then it can lead to stack overflow.
- Random accessing is not possible in stack.
- If the stack falls outside the memory it can lead to abnormal termination.

17. How to represent Queue? Explain it?

Ans-A queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.



18. Briefly explain working of Queue.
Ans-Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Operations on Queue-

- **enqueue()** – add (store) an item to the queue.
- **dequeue()** – remove (access) an item from the queue.
- **peek()** – Gets the element at the front of the queue without removing it.
- **isfull()** – Checks if the queue is full.
- **isempty()** – Checks if the queue is empty.

19. Explain how to perform operations on Queue.
Ans-Working of Queue
Queue operations work as follows:

- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last element of the queue
- initially, set value of FRONT and REAR to -1

Enqueue Operation

- check if the queue is full
- for the first element, set the value of FRONT to 0
- increase the REAR index by 1
- add the new element in the position pointed to by REAR

Dequeue Operation

- check if the queue is empty
- return the value pointed by FRONT
- increase the FRONT index by 1

20. List and explain advantages and disadvantages of Queue.
Ans-Advantages of Queue:

- A large amount of data can be managed efficiently with ease.
- Queues are useful when a particular service is used by multiple consumers.
- Queues are fast in speed for data inter-process communication.
- Queues can be used in the implementation of other data structures.

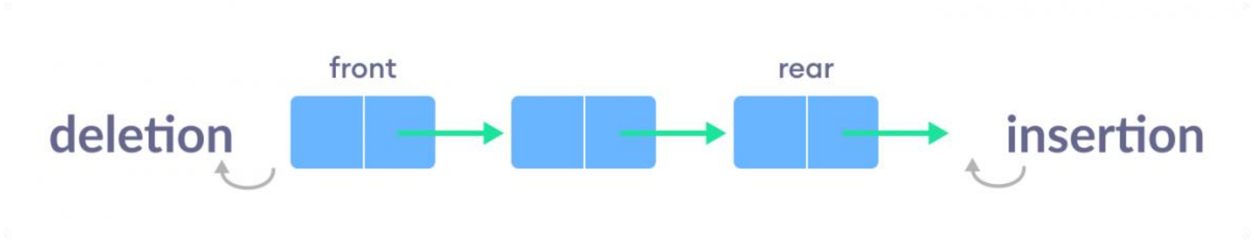
Disadvantages of Queue:

- The operations such as insertion and deletion of elements from the middle are time consuming.
- Limited Space.
- Searching an element takes O(N) time.
- Maximum size of a queue must be defined prior.

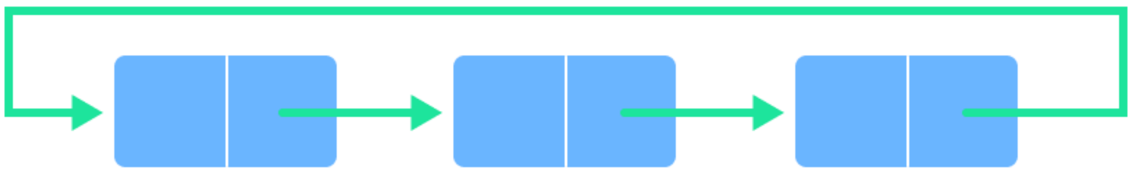
21. Explain types of Queue.
Ans-There are four different types of queues:

- Simple Queue
- Circular Queue
- Priority Queue
- Double Ended Queue

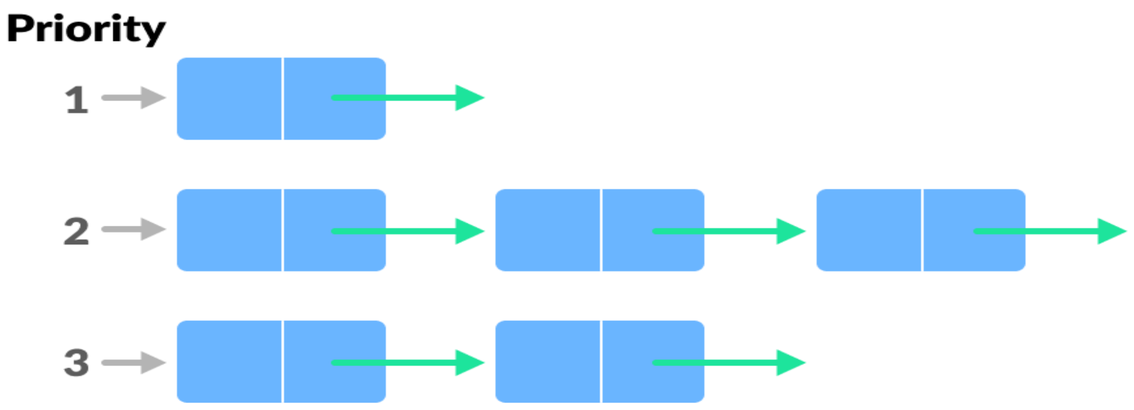
Simple Queue-In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



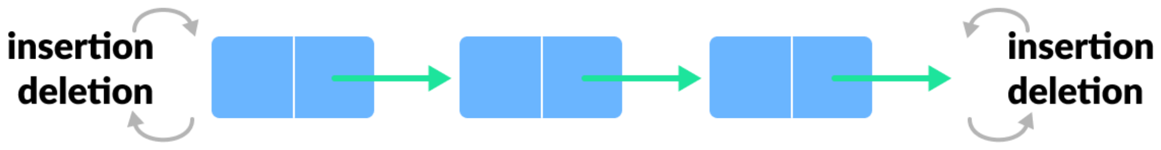
Circular Queue-In a circular queue, the last element points to the first element making a circular link.



Priority Queue-A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority.



Doubly Ended Queue-In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.

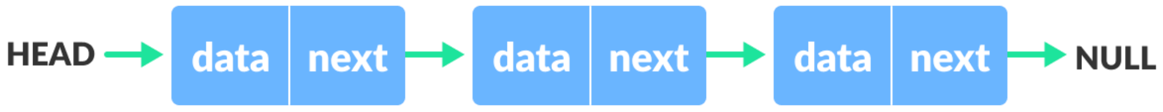


22. Explain types of linked list.

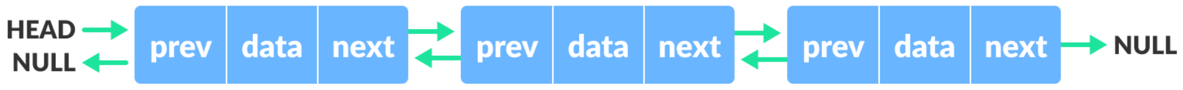
Ans-There are three common types of Linked List.

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

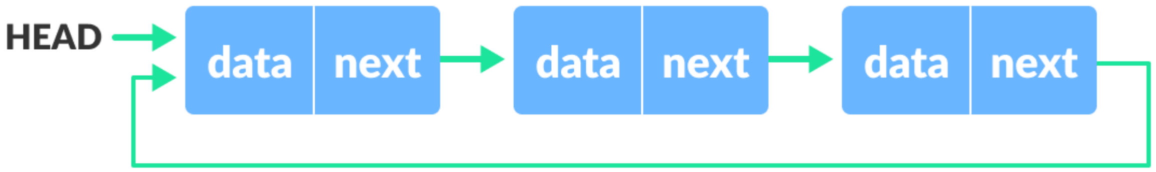
Singly Linked List-It is the most common. Each node has data and a pointer to the next node.



Doubly Linked List-We add a pointer to the previous node in a doubly-linked list. Thus, we can go in either direction: forward or backward.



Circular Linked List-A circular linked list is a variation of a linked list in which the last element is linked to the first element. This forms a circular loop.



23. Explain functions of Stack.

Ans-

- push() to insert an element into the stack
- pop() to remove an element from the stack
- top() Returns the top element of the stack.
- isEmpty() returns true is stack is empty else false
- size() returns the size of stack

24. Explain functions of queue.

Ans-

- enqueue(): Inserts an element at the end of the queue i.e. at the rear end.
- dequeue(): This operation removes and returns an element that is at the front end of the queue.
- front(): This operation returns the element at the front end without removing it.
- rear(): This operation returns the element at the rear end without removing it.
- Empty(): This operation indicates whether the queue is empty or not.
- size(): This operation returns the size of the queue i.e. the total number of elements it contains.

25. Explain benefits of ADT.
Ans-
Benefits of using Abstract Data Types

- Code is easier to understand (e.g., it is easier to see "high-level" steps being performed, not obscured by low-level code).
- Implementations of ADTs can be changed (e.g., for efficiency) without requiring changes to the program that uses the ADTs.
- ADTs can be reused in future programs.

26. Explain LIST ADT in detail.
Ans-
List ADT
The data is generally stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list.
The data node contains the pointer to a data structure and a self-referential pointer which points to the next node in the list.
The List ADT Functions is given below:

- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.
- replace() – Replace an element at any position by another element.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise return false.
- isFull() – Return true if the list is full, otherwise return false.

27. Explain prefix to postfix conversion.
Ans-Prefix: An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).
Example : *+AB-CD (Infix : (A+B) * (C-D))

Postfix: An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).
Example : AB+CD-* (Infix : (A+B * (C-D))

Input : Prefix : *+AB-CD
Output : Postfix : AB+CD-*
Explanation : Prefix to Infix : (A+B) * (C-D)
Infix to Postfix : AB+CD-*

Input : Prefix : *-A/BC-/AKL
Output : Postfix : ABC/-AK/L-*
Explanation : Prefix to Infix : (A-(B/C))*((A/K)-L)
Infix to Postfix : ABC/-AK/L-*