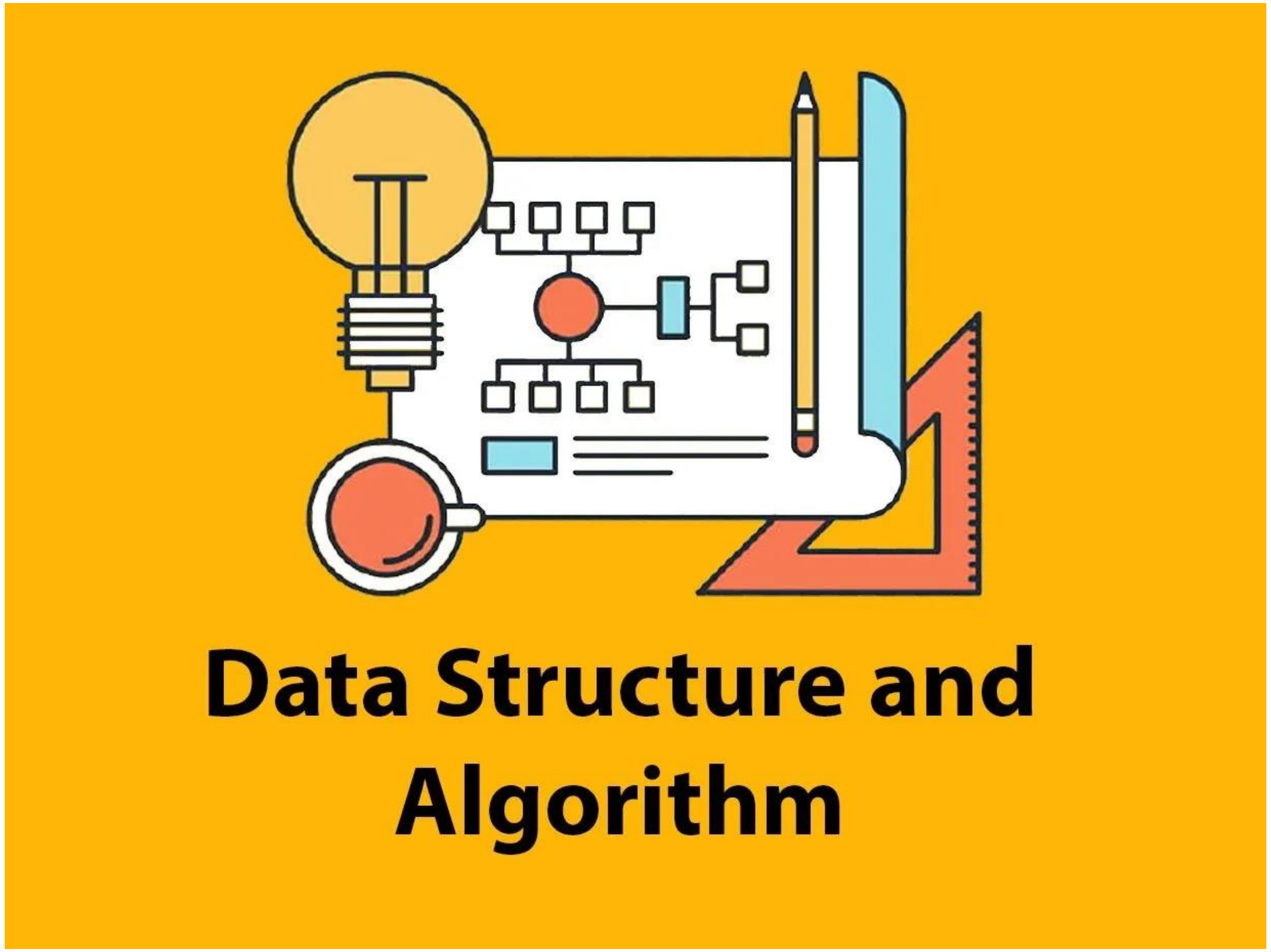

Data Structure and Algorithm

Programming and data structure

- Data, Entity, Information, Difference between Data and Information, Data type , Build in data type, Abstract data type, Definition of data structures, Types of Data Structures: Linear and Non-Linear Data Structure, Introduction to Algorithms: Definition of Algorithms, Difference between algorithm and programs, properties of algorithm, Algorithm Design Techniques, Performance Analysis of Algorithms, Complexity of various code structures, Order of Growth, Asymptotic Notations.

Programming and data structure

- Definition, Single and Multidimensional Arrays, Representation of Arrays: Row Major Order, and Column Major Order, Derivation of Index Formulae for 1-D,2-D Array Application of arrays, Sparse Matrices and their representations. Recursion: recursion in C, example of recursion, Tower of Hanoi Problem, simulating recursion, Backtracking,, recursive algorithms, principles of recursion.

Programming and data structure

- Array Implementation and Pointer Implementation of Singly Linked Lists, Doubly Linked List, Circularly Linked List, Operations on a Linked List. Insertion, Deletion, Traversal, Polynomial Representation and Addition Subtraction & Multiplications of Single variable.

- Abstract Data Type, Primitive Stack operations: Push & Pop, Array and Linked Implementation of Stack in C, Application of stack: Prefix and Postfix Expressions, Evaluation of postfix expression, Iteration and RecursionPrinciples of recursion, Tail recursion, Removal of recursion Problem

Programming and data structure

- solving using iteration and recursion with examples such as binary search, Fibonacci numbers, and Hanoi towers.
- Operations on Queue: Create, Add, Delete, Full and Empty, Circular queues, Array and linked implementation of queues in C, Dequeue and Priority Queue.
- Concept of Searching, Sequential search, Index Sequential Search, Binary Search. Concept of Hashing & Collision resolution Techniques used in Hashing.

Programming and data structure

- Insertion Sort, Selection Sort, Bubble Sort, Heap Sort, Comparison of Sorting Algorithms, Sorting in Linear Time: Counting Sort and Bucket Sort.
- Terminology used with Graph, Data Structure for Graph Representations: Adjacency Matrices, Adjacency List, Adjacency. Graph Traversal: Depth First Search and Breadth First Search, Connected Component.

Programming and data structure

- Basic terminology used with Tree, Binary Trees, Binary Tree Representation: Array Representation and Pointer (Linked List) Representation, Binary Search Tree, Complete Binary Tree, A Extended Binary Trees, Tree Traversal algorithms: Inorder, Preorder and Postorder, Constructing Binary Tree from given Tree Traversal, Operation of Insertion, Deletion, Searching & Modification of data in Binary Search Tree. Threaded Binary trees, Huffman coding using Binary Tree, AVL Tree and B Tree

Algorithms

- Algorithm Analysis, Time Space Tradeoff, Asymptotic Notations, Conditional asymptotic notation, Removing condition from the conditional asymptotic notation, Properties of big-Oh notation.
- Recurrence equations, Solving recurrence equations, Analysis of linear search, Divide and Conquer: General Method, Binary Search, Finding Maximum and Minimum, Merge Sort.
- General Method, Multistage Graphs, All-Pair shortest paths, Optimal binary search trees.
- General Method, 8-Queens problem, Hamiltonian problem. Connected Components, Spanning Trees, Biconnected components,
- Introduction to NP Hard and NP-Completeness.

What is Data Structures…???

- A data structure is a data organization, management, and storage format that enables efficient access and modification.
- More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.
- The idea is to reduce the space and time complexities of different tasks.

Why to Learn Data Structure and Algorithms?

- As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

- Data Search − Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.

Applications of Data Structure and Algorithms

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

Why to Learn Data Structure and Algorithms?

- Processor speed − Processor speed although being very high, falls limited if the data grows to billion records.
- Multiple requests − As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

Types

- ✓ Array
- ✓ Linked List
- ✓ Stack
- ✓ Queue
- ✓ Binary Tree
- ✓ Binary Search Tree
- ✓ Heap
- ✓ Hashing
- ✓ Graph

Major Operations

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

Linear Data Structures

A Linear data structure have data elements arranged in sequential manner and each member element is connected to its previous and next element.

Linear Data Structures

1. Array
2. Stack
3. Queue
4. Linked List

Array

- ✓ Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
- ✓ The size of an array must be provided before storing data.

Array

- ✓ The array is a data structure used to store homogeneous elements at contiguous locations.
- ✓ Array is the simplest data structure where each data element can be randomly accessed by using its index number.

Properties of the Array

1. Each element is of same data type and carries a same size i.e. int = 4 bytes.
2. Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
3. Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of data element.

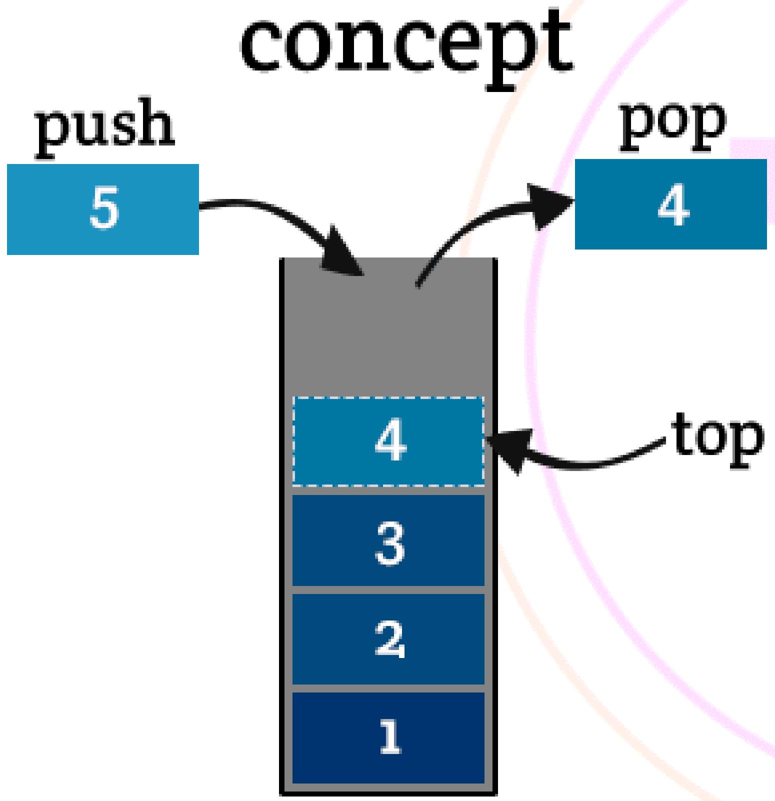
Stack

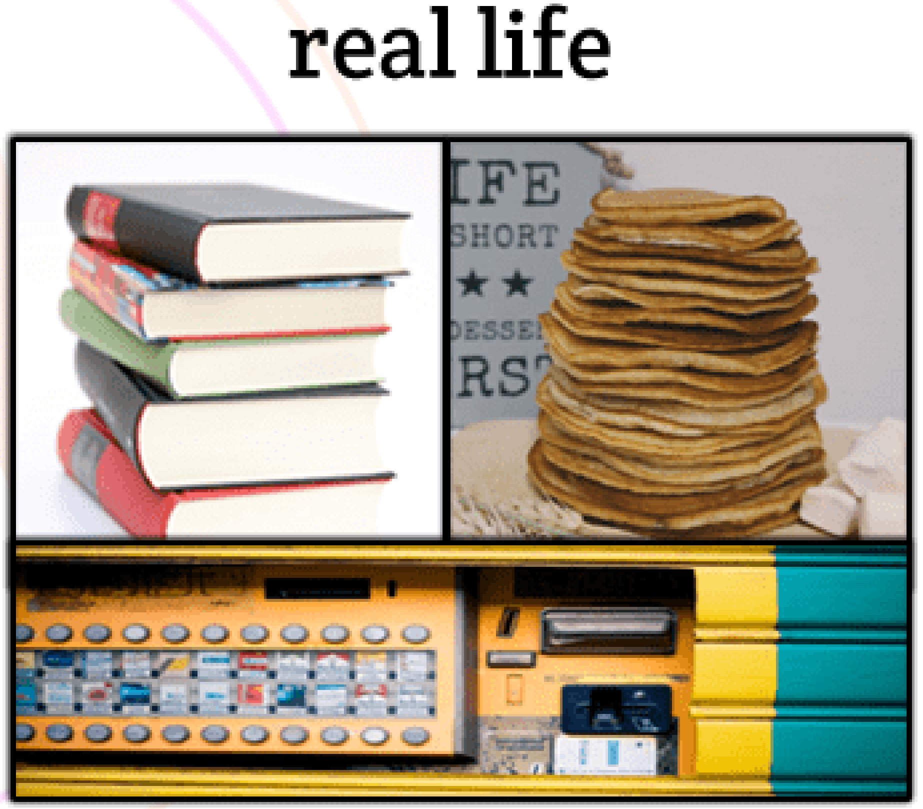
A stack or LIFO (last in, first out) is an abstract data type that serves as a collection of elements, with two principal operations:

- **push**, which adds an element to the collection.
- **pop**, which removes the last element that was added.

In stack both the operations of push and pop take place at the same end that is top of the stack.

Stack



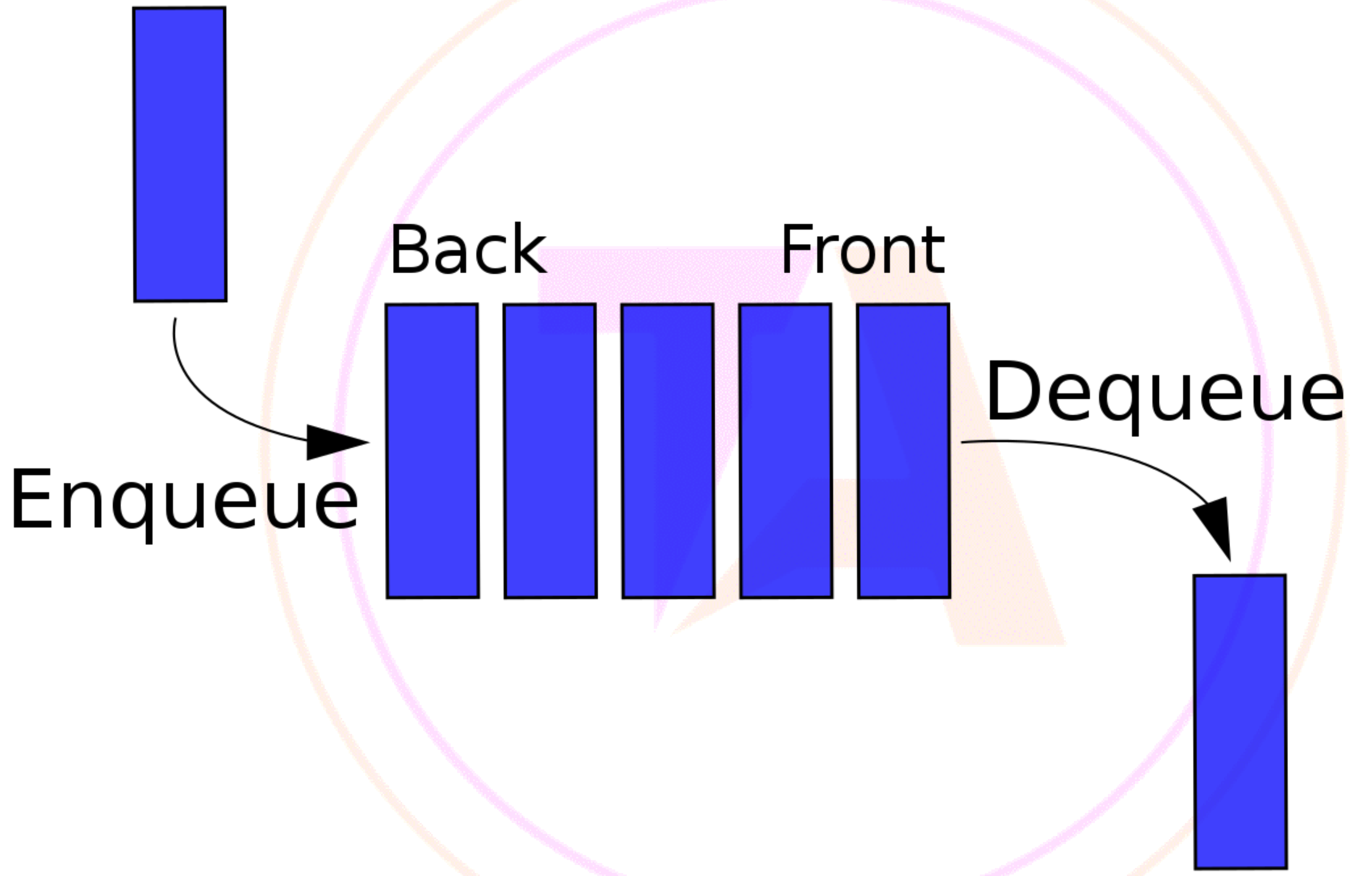


last-in-first-out (LIFO)

Queue

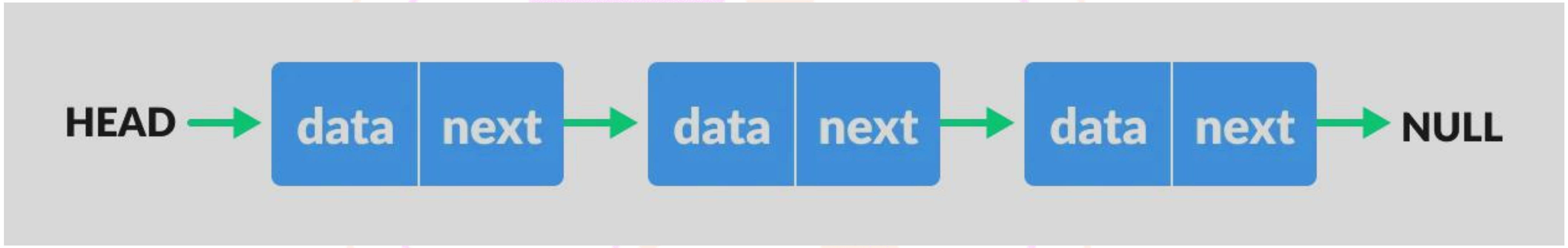
A queue or FIFO (first in, first out) is an abstract data type that serves as a collection of elements, with two principal operations:

Enqueue, the process of adding an element to the collection. (The element is added from the rear side)
Dequeue the process of removing the first element that was added. (The element is removed from the front side).


Back
Front
Enqueue
Dequeue

Linked List

- ✓ Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.
- ✓ A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- ✓ The last node of the list contains pointer to the null.

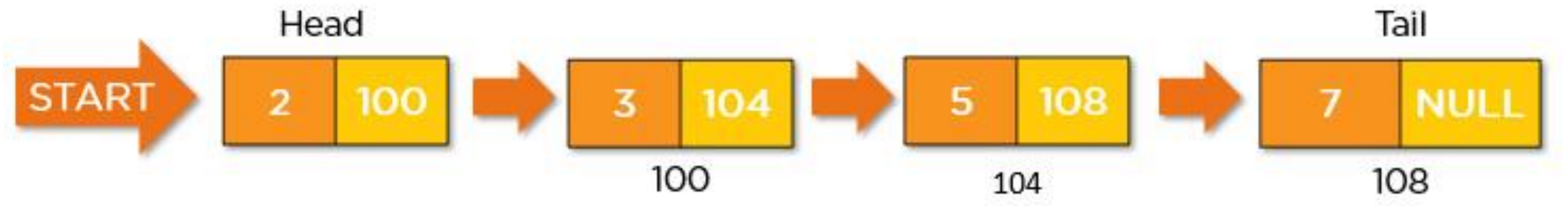

HEAD
data
next
data
next
data
next
NULL

Types Of Linked List

- ✓ Singly linked lists
- ✓ Doubly linked lists
- ✓ Circular linked lists
- ✓ Circular doubly linked lists

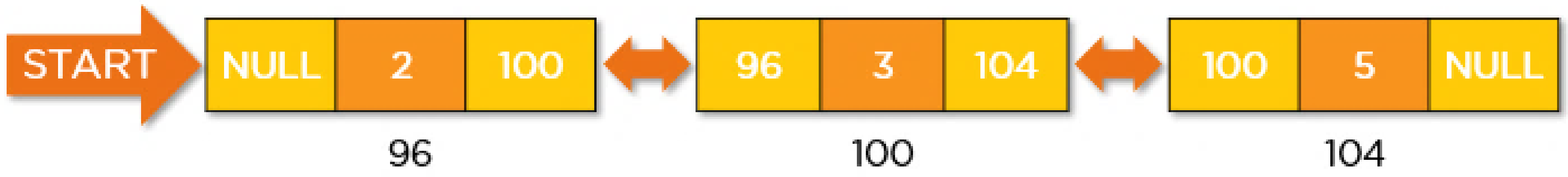
Singly Linked List

A singly linked list is a unidirectional linked list. So, you can only traverse it in one direction, i.e., from head node to tail node

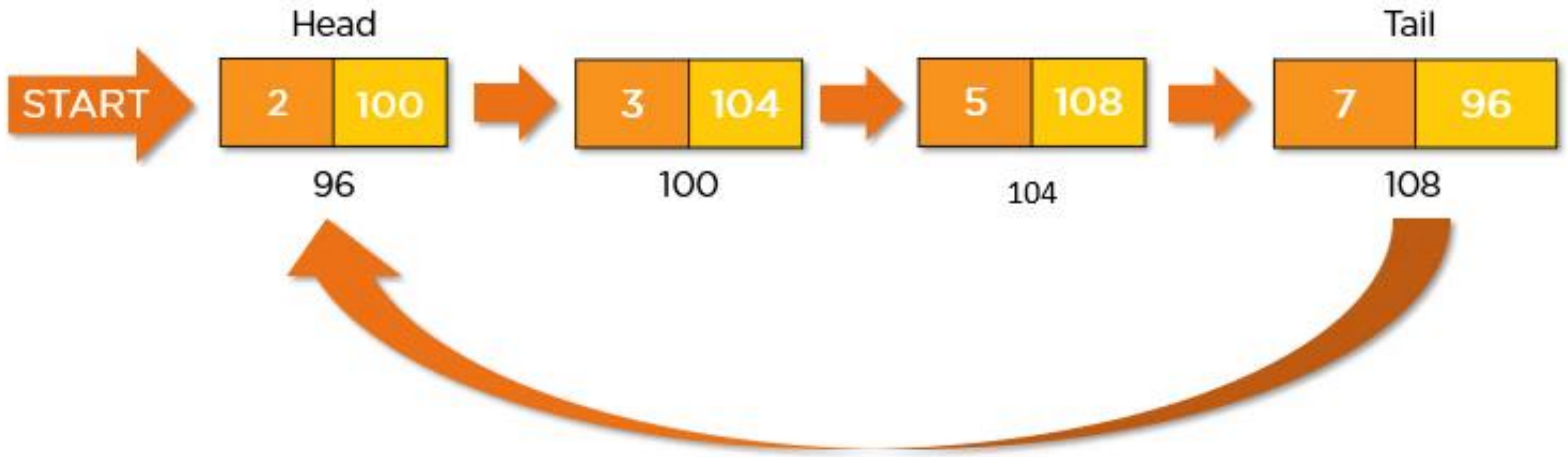


Doubly Linked List

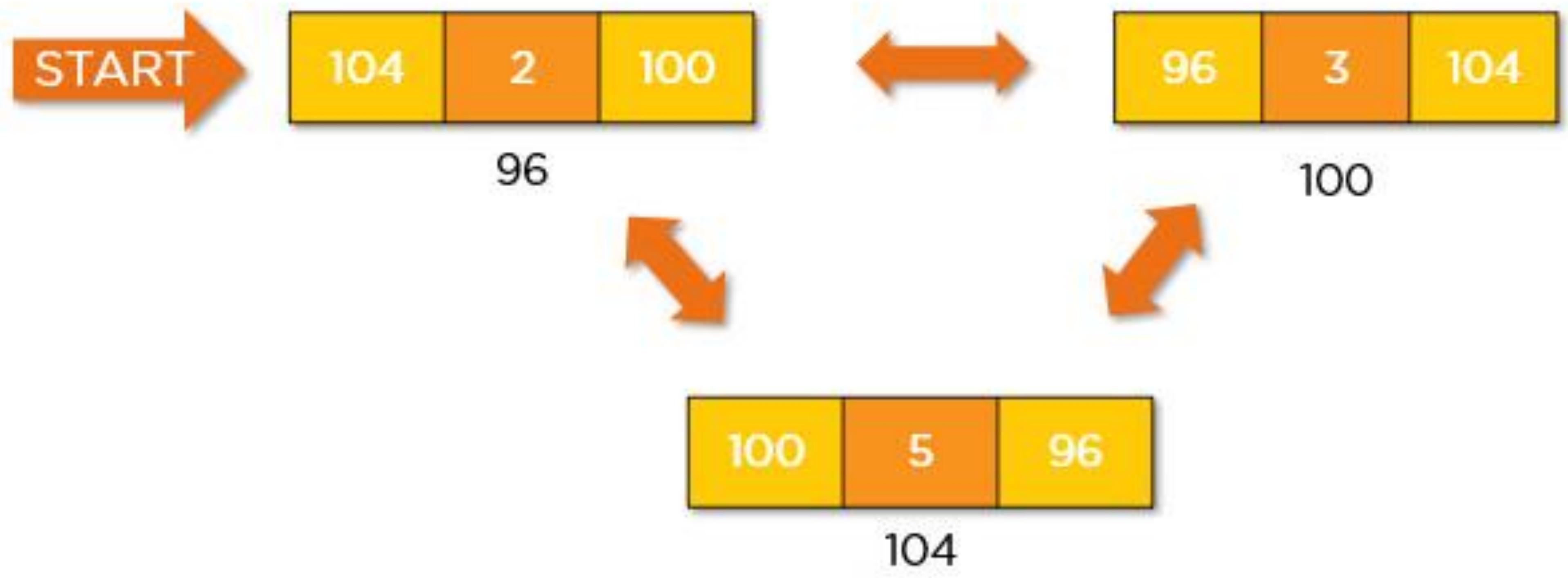
A doubly linked list is a bi-directional linked list. So, you can traverse it in both directions. Unlike singly linked lists, its nodes contain one extra pointer called the previous pointer. This pointer points to the previous node.



Circular Linked List



Circular Doubly Linked List



Queue:

A Queue is a linear structure which follows a particular order in which the operations are performed.
The order is First In First Out (FIFO).
A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first.

There are different types of queues which are used in different scenarios. They are:

Circular Queue:

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called ‘Ring Buffer’. This queue is primarily used in the following cases:

Traffic system: In a computer-controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.

CPU Scheduling: Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

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.

There are two types of Priority Queues. They are:

Ascending Priority Queue

Descending priority Queue

Ascending Priority Queue: Element can be inserted arbitrarily but only smallest element can be removed.
For example, suppose there is an array having elements 4, 2, 8 in the same order. So, while inserting the elements, the insertion will be in the same sequence but while deleting, the order will be 2, 4, 8.

Descending priority Queue: Element can be inserted arbitrarily but only the largest element can be removed first from the given Queue. For example, suppose there is an array having elements 4, 2, 8 in the same order. So, while inserting the elements, the insertion will be in the same sequence but while deleting, the order will be 8, 4, 2.

The priority queue is primarily used to implement the CPU scheduling algorithms.

Q. Queue size = 5

Insert (15)
Insert (25)
Delete
Insert (35)
Delete
Delete
Delete

Applications of Queue

1. CPU scheduling, Disk Scheduling
2. When data is transferred asynchronously between two processes. The queue is used for synchronization. For example: IO Buffers, pipes, file IO, etc
3. Handling of interrupts in real-time systems.
4. Call Center phone systems use Queues to hold people calling them in order.

To perform enqueue and dequeue efficiently on a queue, following operations are also required:

- **IS EMPTY :** used to check whether the queue has any element or not, so as to avoid Underflow exception while performing dequeue operation.

- **PEEK :** used to view elements at the front of the queue, without removing it from the queue.

- **IS FULL :** used to check whether any more element can be added to the queue or not, to avoid Overflow exceptions while performing enqueue operation.

Show the status of deque after each operation

peek()
insertFront(12)
insertRear(67)
deletionFront()
insertRear(43)
deletionRear()
deletionFront()
deletionRear()

Application of STACK

- When we need to reverse a string, the string is traversed from the last character till the first character. i.e. characters are traversed in the reverse order of their appearance in the string. This is very easily done by putting the characters of a string in a stack.

Application of STACK

- We use text/image editor for editing the text/image where we have options to redo/undo the editing done.

When we click on the redo /undo icon, the most recent editing is redone/undone.

Application of STACK

- While browsing the web, we move from one web page to another by accessing links between them. In order to go back to the last visited web page, we may use the back button on the browser.

Application of STACK

Evaluation of Arithmetic Expressions

Q. If the elements “A”, “B”, “C” and “D” are placed in a stack and are deleted one at a time, what is the order of removal?

a) ABCD
b) DCBA
c) DCAB
d) ABDC

Q. In a stack, if a user tries to remove an element from an empty stack it is called __________

a) Underflow
b) Empty collection
c) Overflow
d) Garbage Collection

Q. Pushing an element into stack already having five elements and stack size of 5, then stack becomes ____________

a) Overflow
b) Crash
c) Underflow
d) User flow

Q. Maximum number of parenthesis that will appear on the stack at any time for below parenthesis expression.

(( ) (( )) ((( ))))

a) 3
b) 4
c) 5
d) 2

The way to write arithmetic expression is known as a **notation**.

An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the output of an expression.

These notations are −

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

Order of Precedence and Associativity

Sr.No.	Operator	Precedence	Associativity
1	Exponentiation ^	Highest	Right Associative
2	Multiplication (∗) & Division (/)	Second Highest	Left Associative
3	Addition (+) & Subtraction (−)	Lowest	Left Associative

1. Infix Notation:

We write expression in **infix** notation, e.g. a - b + c, where operators are used **in**-between operands.

2. Prefix Notation:

In this notation, operator is **prefix**ed to operands, i.e. operator is written ahead of operands.

Example, **+ab**. This is equivalent to its infix notation **a + b**. Prefix notation is also known as **Polish Notation**.

3. Postfix Notation:

This notation style is known as **Reversed Polish Notation**. In this notation style, the operator is **postfixed** to the operands i.e., the operator is written after the operands.

Example: **a + b,** Postfix notation is **ab+.**

Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

The value of postfix expression:
8 3 4 + - 3 8 2 / + * 2 $ 3+ is

(A) 17	(B) 131
(C) 64	(D) 52

What is the value of the postfix expression?
a b c d + - * (where a = 8, b = 4, c = 2 and d = 5)

(A) -3/8 (B) -8/3
(C) 24 (D) -24

The postfix expression AB+CD–* can be evaluated using a

(A) stack	(B) tree
(C) queue	(D) linked list

Given the following prefix expression:

$* + 3 + 3 \uparrow 3 + 3\ 3\ 3$

What is the value of the prefix expression?

(A) 2178 (B) 2199

(C) 2205 (D) 2232

Consider the following postfix expression with single digit operands:

623∗/42∗+68∗−

The top two elements of the stack after the second ∗ is evaluated, are:

(1) 8,2 (2) 8,1 (3) 6,2 (4) 6,3

The following postfix expression is evaluated using a stack
823^/23* + 51* –
The top two elements of the stack after first * is evaluated
(A) 6, 1 (B) 5, 7 (C) 3, 2 (D) 1, 5

The seven elements A, B, C, D, E, F and G are pushed onto a stack in reverse order, i.e., starting from G. The stack is popped five times and each element is inserted into a queue. Two elements are deleted from the queue and pushed back onto the stack. Now, one element is popped from the stack. The popped item is ....

(1) A	(2) B
(3) F	(4) G

At a hill station, the parking lot is one long drive way snaking up a hill side. Cars drive in and park right behind the car in front of them, one behind another. A car can't leave until all the cars in front of it have left. Is the parking lot more like

(A) An array (B) A stack

(C) A queue (D) A linked list