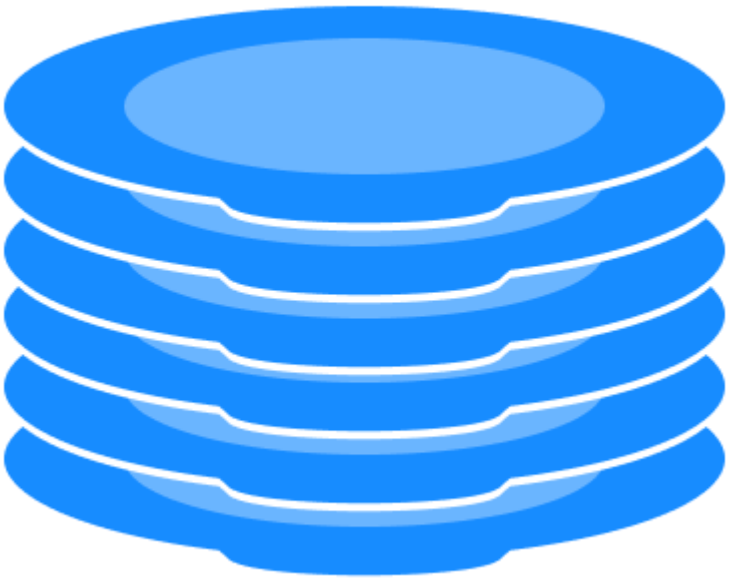
Stack Data Structure

A stack is a **linear data structure that follows** the principle of **Last in First Out (LIFO)**. This means the last element inserted inside the stack is removed first. You can think of the stack data structure as the pile of plates on top of another.



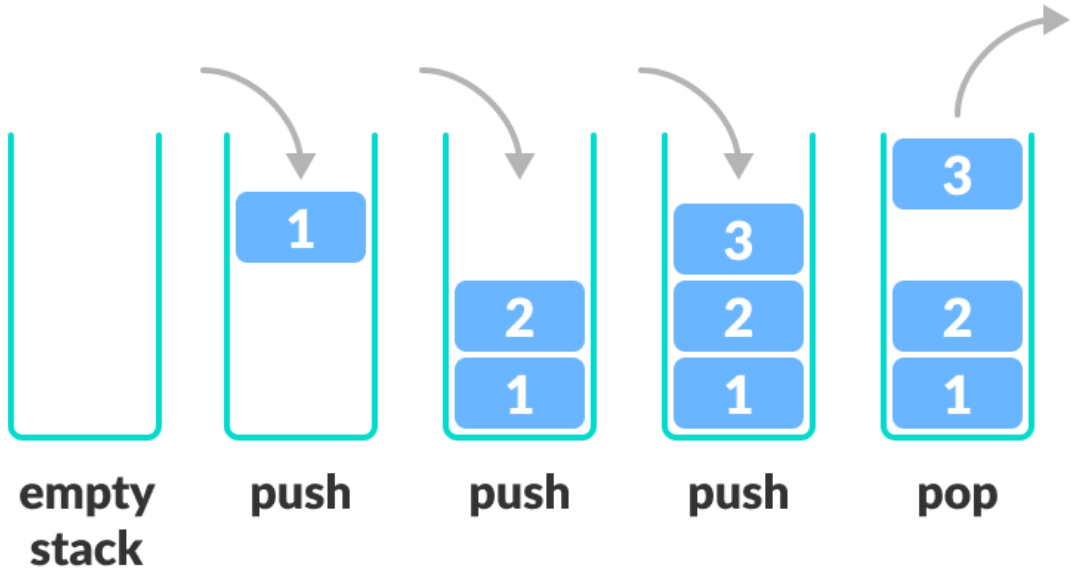
Stack representation similar to a pile of plate
Here, you can:

- **Put a new plate on top**
- **Remove the top plate**

And, if you want the plate at the bottom, you must first remove all the plates on top. This is exactly how the stack data structure works.

LIFO Principle of Stack

In programming terms, **putting an item on top of the stack** is called **push** and **removing an item** is called **pop**.



Stack Push and Pop Operations

In the above image, although item **3** was kept last, it was removed first. This is exactly how the **LIFO (Last In First Out) Principle** works.

Basic Operations of Stack

There are some basic operations that allow us to perform different actions on a stack.

- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty

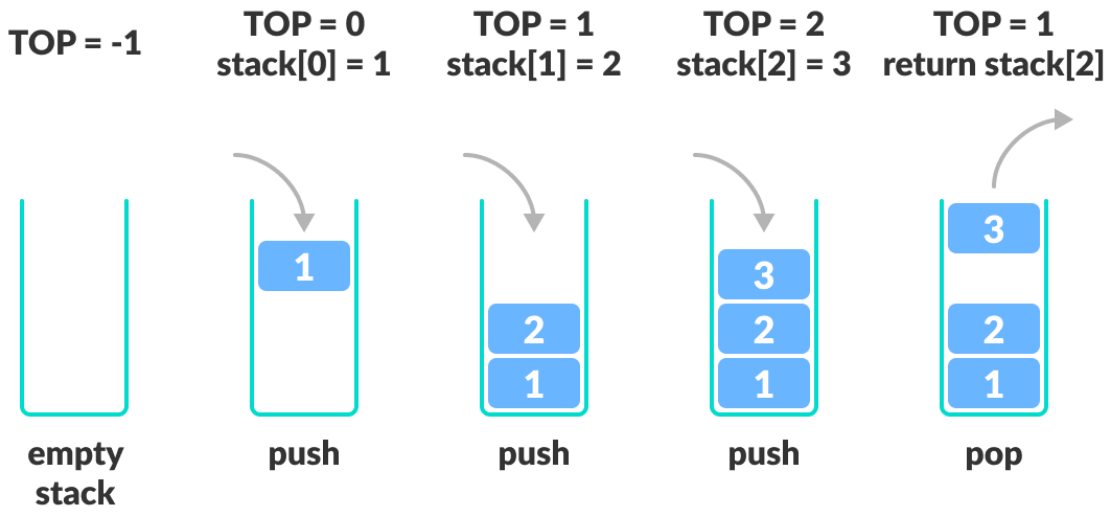
- **IsFull**: Check if the stack is full
- **Peek**: Get the value of the top element without removing it

---

Working of Stack Data Structure

The operations work as follows:

1. **A pointer called `TOP` is used to keep track of the top element in the stack.**
2. When initializing the stack, **we set its value to -1 so that we can check if the stack is empty by comparing `TOP == -1`.**
3. On pushing an element, **we increase the value of `TOP` and place the new element in the position pointed to by `TOP`.**
4. On popping an element, **we return the element pointed to by `TOP` and reduce its value.**
5. **Before pushing, we check if the stack is already full**
6. **Before popping, we check if the stack is already empty**

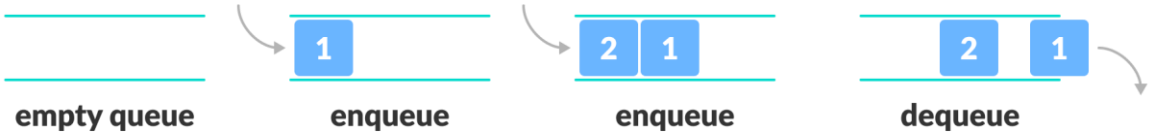


Working of Stack Data Structure

Queue Data Structure

A queue is a useful data structure in programming. **It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.**

Queue follows the **First In First Out (FIFO)** rule - the item that goes in first is the item that comes out first.



FIFO Representation of Queue

In the above image, since 1 was kept in the queue before 2, it is the first to be removed from the queue as well. It follows the **FIFO** rule.

In programming terms, **putting items in the queue** is called **enqueue**, and **removing items from the queue** is called **dequeue**.

Basic Operations of Queue

A queue is an **object (an abstract data structure - ADT)** that allows the following operations:

- **Enqueue**: Add an element to the end of the queue
- **Dequeue**: Remove an element from the front of the queue
- **IsEmpty**: Check if the queue is empty
- **IsFull**: Check if the queue is full
- **Peek**: Get the value of the front of the queue without removing it

---

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**

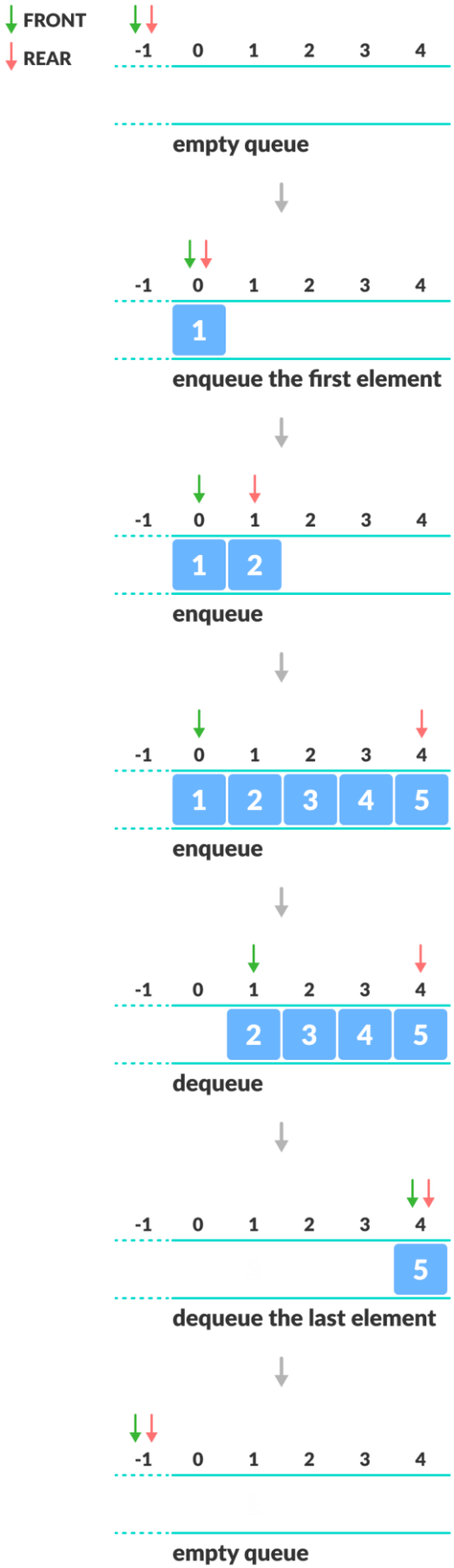
Enqueue Operation

- 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**
- for the last element, reset the values of **FRONT and REAR to -1**


FRONT
REAR
-1 0 1 2 3 4
empty queue
1
enqueue the first element
2
enqueue
3 4 5
enqueue
dequeue
dequeue the last element
empty queue

Enqueue and Dequeue Operations

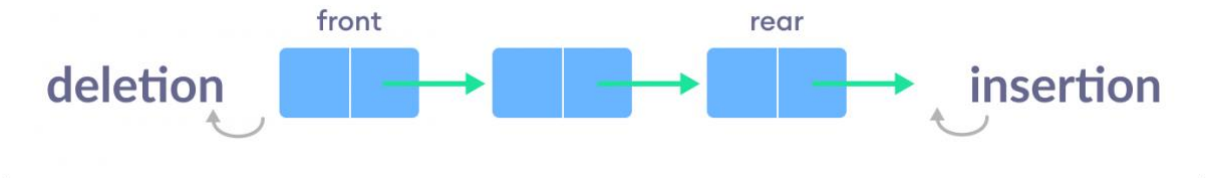
Types of Queues

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.
There are four different types of queues:

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

Simple Queue

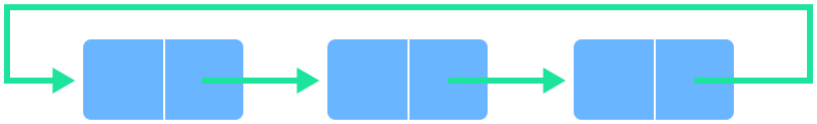
In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



Simple Queue Representation
To learn more, visit Queue Data Structure.

Circular Queue

In a circular queue, **the last element points to the first element making a circular link.**



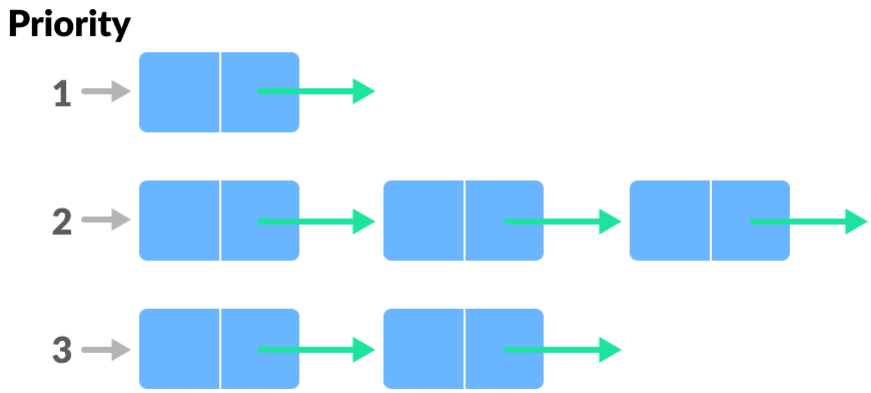
Circular Queue Representation

The main advantage of a circular queue over a simple queue is **better memory utilization**. **If the last position is full and the first position is empty, we can insert an element in the first position**. This action is not possible in a simple queue.

To learn more, visit Circular Queue Data Structure.

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. If elements with the same priority occur, they are served according to their order in the queue.



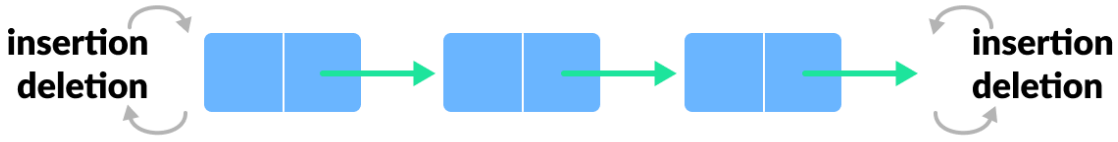
Priority Queue Representation

Insertion occurs based on the arrival of the values and removal occurs based on priority.

To learn more, visit Priority Queue Data Structure.

Deque (Double 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.



Deque Representation