QUEUE

Queue is fundamental data structure that follows the First In First Out (FIFO) principle. Just like standing in a queue at a supermarket, where the person who joins the queue first is the first one to be served, a queue organizes and manages data with an emphasis on the order of access. Queues are crucial in computer science for efficiently handling data and operations, especially when order and timing matter. Adhering to the FIFO principle, queues find application in various programming contexts and algorithmic problem-solving.

Key Methods of Queue:

1. **enqueue:** This method adds an element to the rear of the queue.
2. **dequeue:** This method removes and returns the front element of the queue.
3. **peek:** This method returns the value of the front element without removing it.
4. **count:** This method returns the number of elements in the queue.
5. **isFull:** This method checks if the queue has reached its maximum capacity (if applicable).
6. **isEmpty:** This method checks if the queue is empty.
7. **clear:** This method empties the entire queue, removing all elements from it.

Benefits of FIFO

The FIFO property ensures that the first item added to the queue is the first one to be removed. This property is essential for managing data in scenarios where the order of arrival is important.

Pros

+ **FIFO Behavior:** The First In First Out (FIFO) property of a queue aligns well with various real-world situations and programming tasks, making it intuitive to manage data in the order of arrival.

+ **Simplicity:** Queues are relatively straightforward to grasp and implement. They involve basic operations (enqueue, dequeue, peek), which makes them accessible for beginners.

+ **Efficient Operations:** Enqueuing and dequeuing elements from a queue are both constant-time operations, making them efficient for managing ordered data.

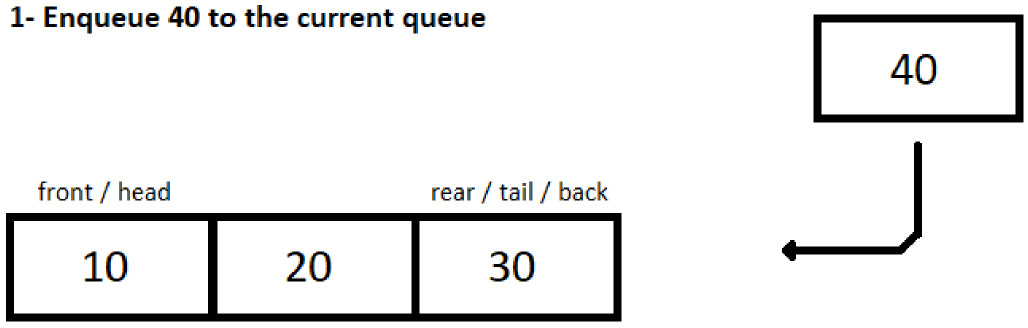
Cons

- **Limited Access:** Queues provide access only to the front and rear elements, restricting interactions with other elements in between.

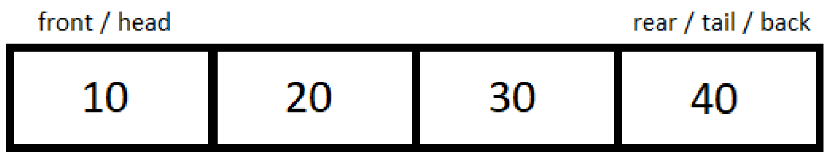
- **Not Universal:** While the FIFO behavior is valuable for many use cases, it may not suit all scenarios. For instance, situations requiring more complex data manipulation or different access patterns might not be well-suited for queues.

The visualization of the usage of key methods is provided below:

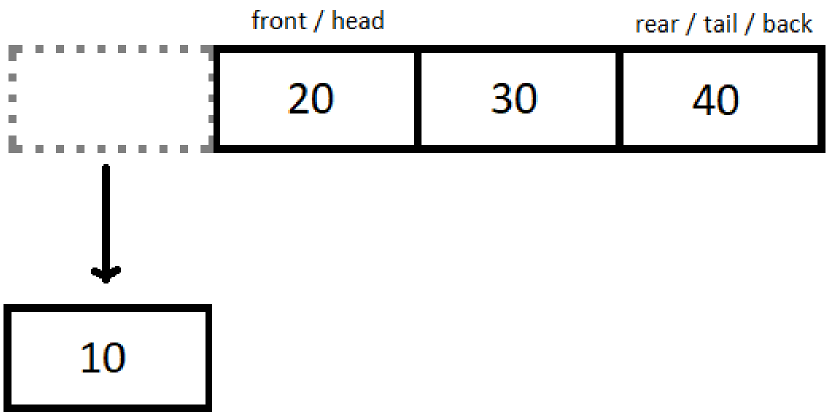
1- Enqueue 40 to the current queue



2- After the insertion of 40



3- Dequeue Operation



IMPORTANT: Remember that when we create an array beforehand, the new item we want to add is placed in an empty index behind "rear" if available, or the "extend" method is called to enlarge the array. In the "dequeue" section, note that since the current "front" element is removed from the array, that index will be empty, and if this continues, the "shrink" method is called to reduce remaining empty spaces and improve performance. If you thoroughly examine the queue code in this repository, you'll understand it better.