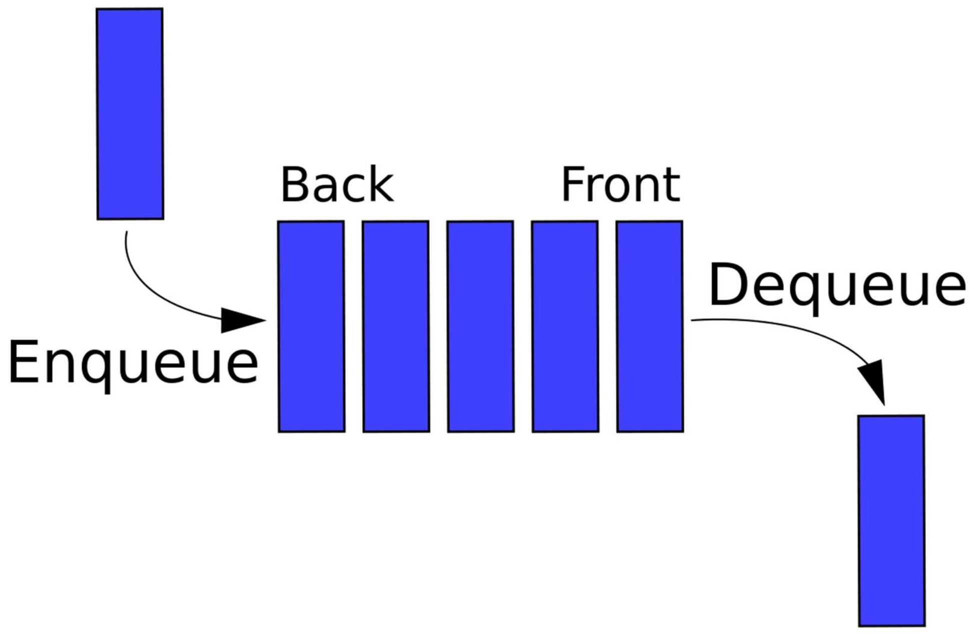
Queues

Introduction

- A **queue** is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence.
- The end of the sequence at which elements are added is called the back, tail, or rear of the queue.
- The end at which elements are removed is called the head or front of the queue

Reading Material: Queue Data Structure



Queue Data Structure - First In First Out

Stacks & Queues - https://youtu.be/wjI1WNcIntg

Applications of a Queue

- When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling
- When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
- **In Operating systems:**
    - **Semaphores**
    - FCFS (first come first serve) scheduling, example: FIFO queue
    - Spooling in printers
    - Buffer for devices like keyboard
- **In Networks:**
    - Queues in routers/switches
    - Mail Queues

References:

Applications of Queue Data Structures

Operations on a Queue with Time Complexity

Operation	Time Complexity	Description
Enqueue (insertion)	O(1)	Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
Dequeue (deletion)	O(1)	Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
Front (Get front)	O(1)	Get the front item from the queue.
Rear (Get Rear)	O(1)	Get the last item from the queue.

Depending on the programming languages, there might also be predefined functions that allow operations like checking finding the size of the queue, etc.
See this for more details on queue operations and their implementation.

Type of Queues

- Simple Queue
- Circular Queues
- Deque or Double Ended Queues
- Priority Queues

Implementing a Queue

In queue, insertion and deletion happen at the opposite ends, so implementation is not as simple as stack.

To implement a simple queue using array, create an array *arr* of size *n* and take two variables front and rear both of which will be initialized to 0 which means the queue is currently empty.

Move rear one index ahead every time there is an element being added to the queue. See enqueue code snippet below for implementation.

To delete from the queue, each element is shifted one index towards front from rear to remove the element at front. See dequeue code snippet below for implementation.

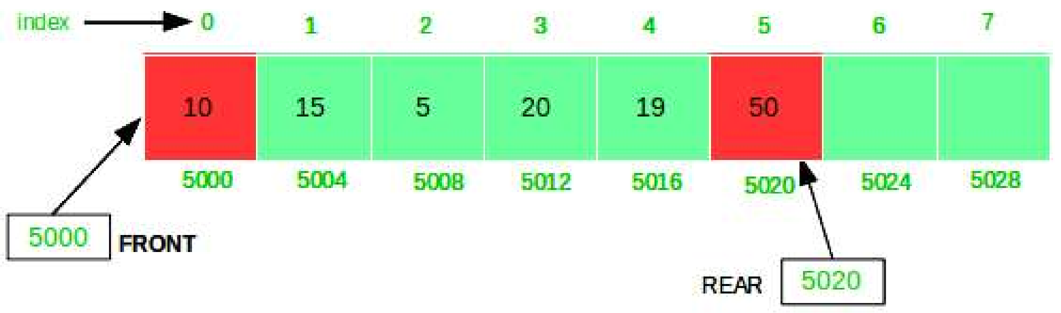
Please refer to **this** link for a complete detailed array implementation of Queues.

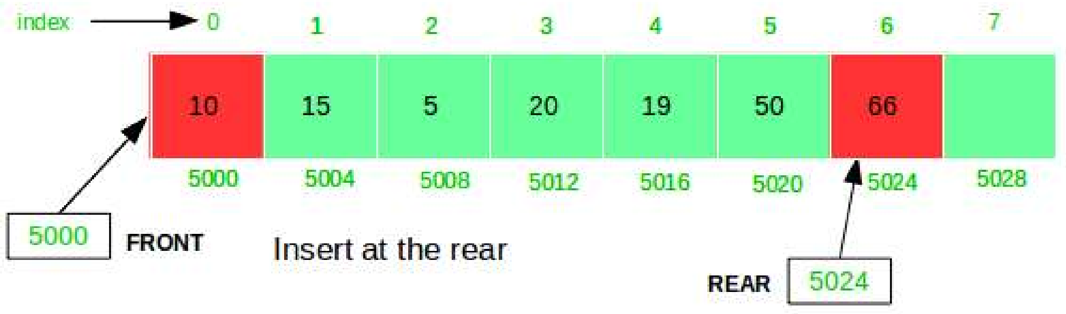
For further implementations in different languages, refer to the links below:

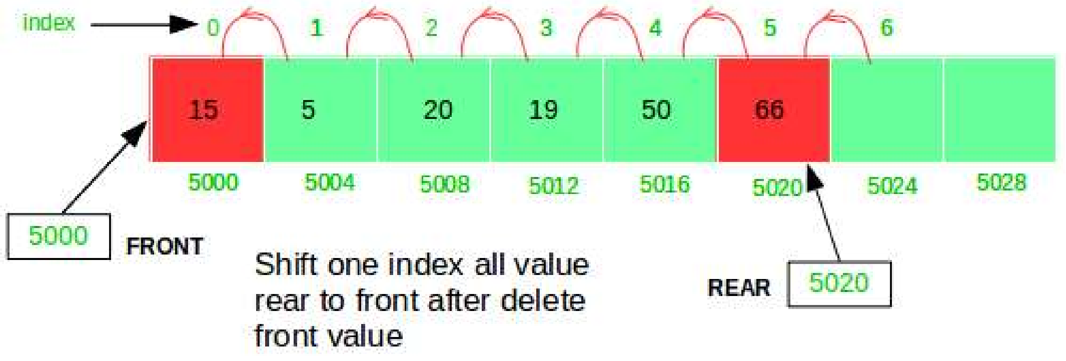
Java Queue Interface

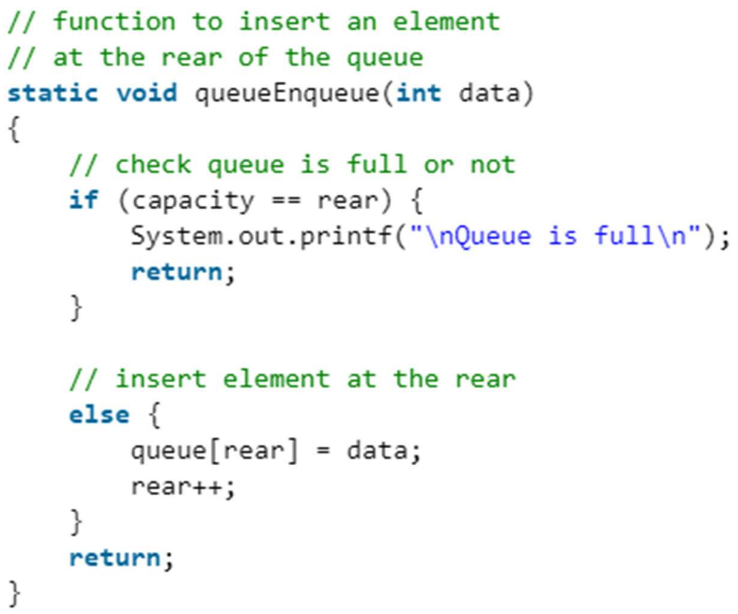
C++ STL

Python





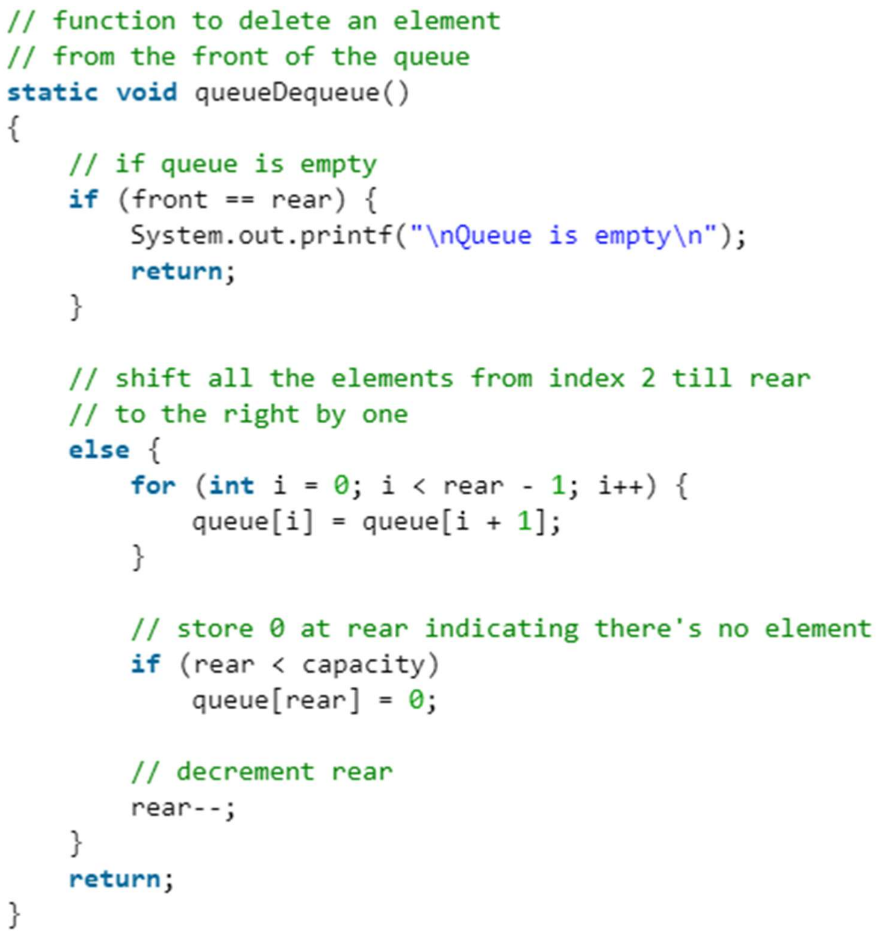




```
// function to insert an element
// at the rear of the queue
static void queueEnqueue(int data)
{
    // check queue is full or not
    if (capacity == rear) {
        System.out.printf("\nQueue is full\n");
        return;
    }

    // insert element at the rear
    else {
        queue[rear] = data;
        rear++;
    }
    return;
}
```

Inserting an element to the Queue



```
// function to delete an element
// from the front of the queue
static void queueDequeue()
{
    // if queue is empty
    if (front == rear) {
        System.out.printf("\nQueue is empty\n");
        return;
    }

    // shift all the elements from index 2 till rear
    // to the right by one
    else {
        for (int i = 0; i < rear - 1; i++) {
            queue[i] = queue[i + 1];
        }

        // store 0 at rear indicating there's no element
        if (rear < capacity)
            queue[rear] = 0;

        // decrement rear
        rear--;
    }
    return;
}
```

Deleting an element to the Queue