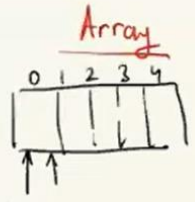
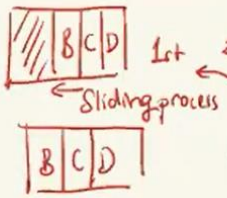
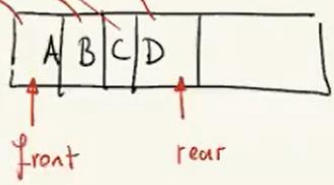
Queue (Kuyruk) Data Structure (Abstract)

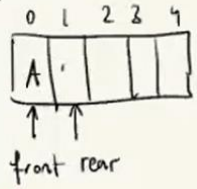
- First In First Out (FIFO)

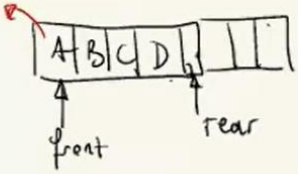




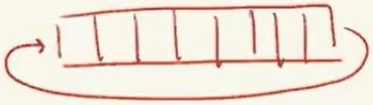


Queue



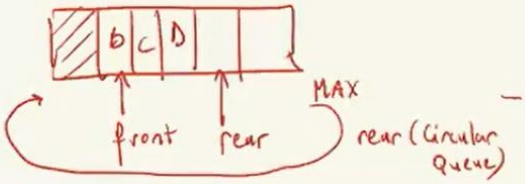


- Circular Queue

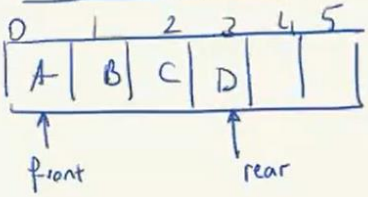


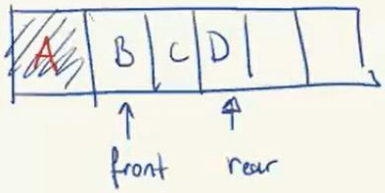
- Priority Queue

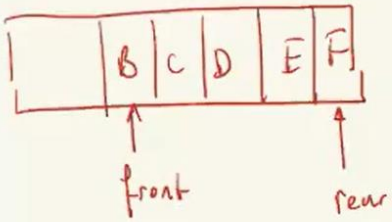
- Double Ended Queue

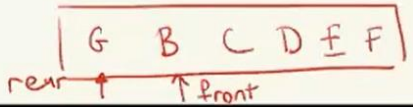


Circular Queue

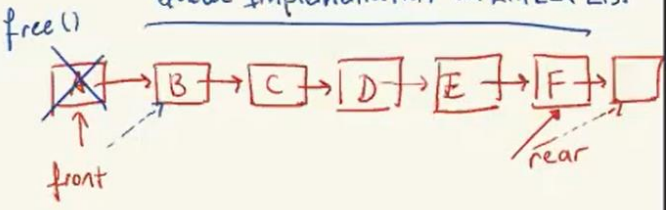








Queue Implementation on Linked List

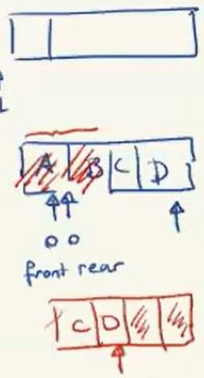


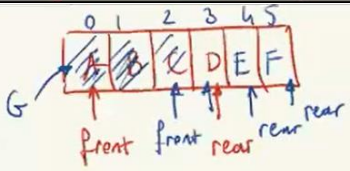
Queue Operations

- isEmpty()
- isFull()
- enQueue()
- deQueue()
- initQueue()

```
int isEmpty() {
    if (size == 0)
        return 1;
    else
        return 0;
}
int isFull() {
    if (size = MAX-1)
        return 1;
    else
        return 0;
}
int enqueue (int val) {
    if (isFull())
        return 0;
    else {
        rear++;
        q[rear] = value;
    }
    return 1;
}
```

```
int isEmpty() {
    if (front == -1)
        return 1;
    else
        return 0;
}
int isFull() {
    if (rear == MAX-1)
        return 1;
    else
        return 0;
}
int dequeue (int *X) {
    if (isEmpty()) {
        return 0;
    else {
        *X = q[front++];
    }
    return 1;
}
```

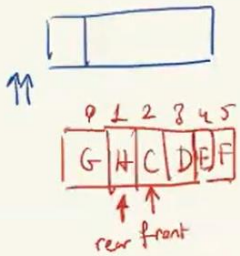


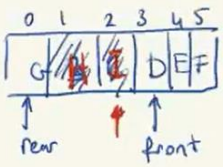


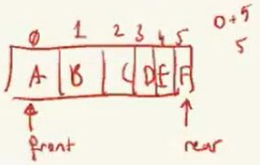
DeQueue A
DeQueue B
EnQueue E
EnQueue F
DeQueue C
EnQueue G

Front: 2
Rear: 5
Front: 3
Rear: 5
Front: 3
Rear: 0

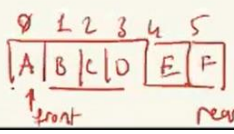
Circular Queue Implementation





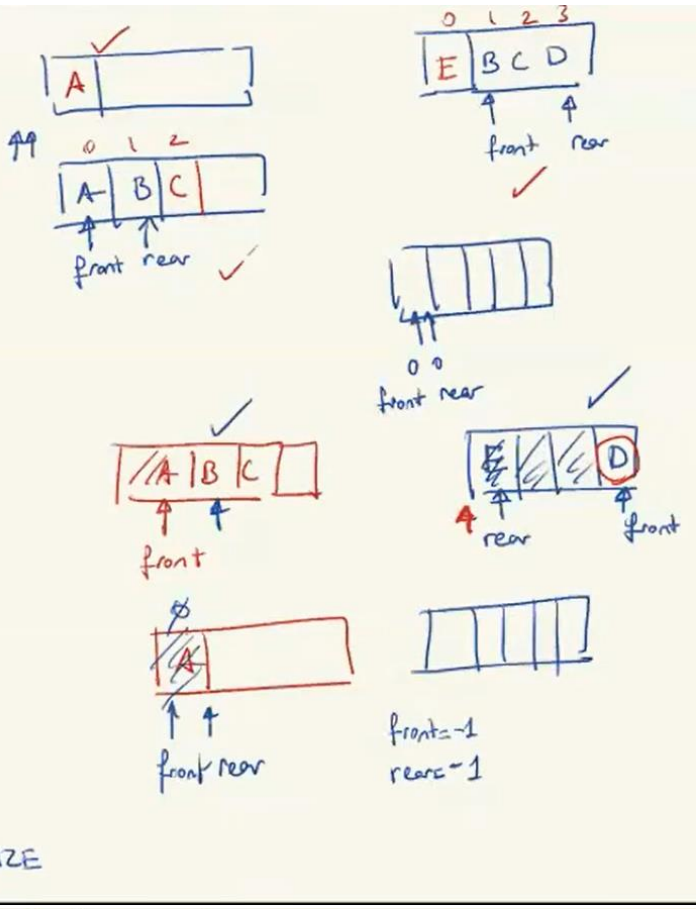



```
void initQueue() {
    int front=-1, rear=-1;
}
int isEmpty() {
    if (front == -1)
        return 1;
    return 0;
}
int isFull() {
    if ((front == rear+1) ||
    (front == 0) && (rear == SIZE-1))
        return 1;
    return 0;
}
```



```
int enQueue (int element){
    if (isFull())
        return 0;
    else { if (front==-1) front=0;
    rear = (rear+1)%SIZE ;
        q[rear]=element;
      return 1;
    }
}

int deQueue (int *X) {
      if (isEmpty()) {
            return 0;
       else{
           *X = q[front];
           if (front==rear){
                    front=-1;
                    rear=-1;
           }
           else
              front=(front+1)% SIZE
      }}      return 1;
```



Comparison Stack versus Queue

	Stack	Queue
Working Principle	LIFO	FIFO
Structure	Same end is used to insert and delete	One end is used for insert Another end is used deletion
Number of pointers	One	Two
Operations	Push and Pop	Enqueue and Dequeue
Variants	It does not have variants	Circular Queue Priority Queue Doubly Ended Queue
Implementation	Simpler	Comparatively Complex