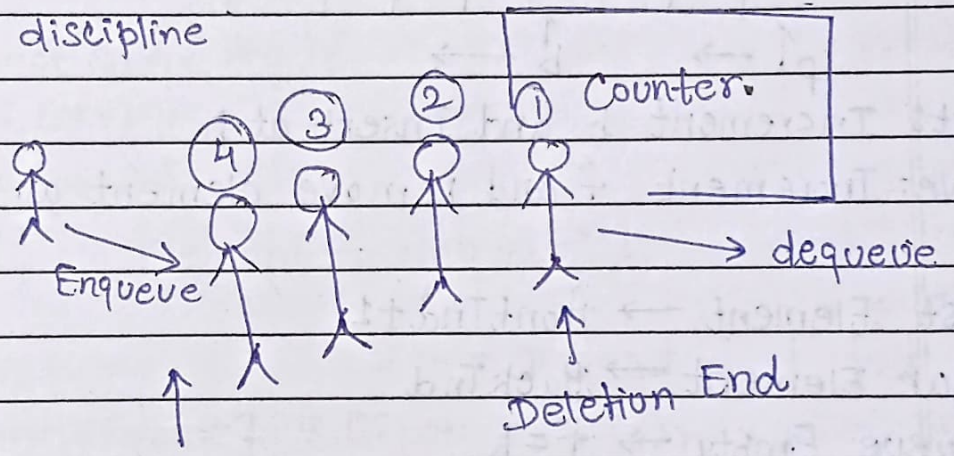
lec-38 Queue Data Structure:

① stack → LIFO discipline

② Queue → FIFO discipline



Queue ADT:

Data: ① Storage
② Ins End
③ Del End

Methods:

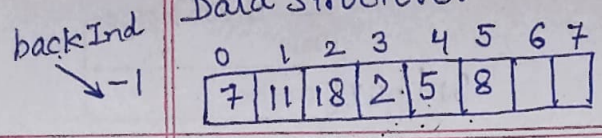
① Enqueue
② dequeue
③ first value
④ last value
⑤ peek(pos)
⑥ IsEmpty
⑦ IsFull

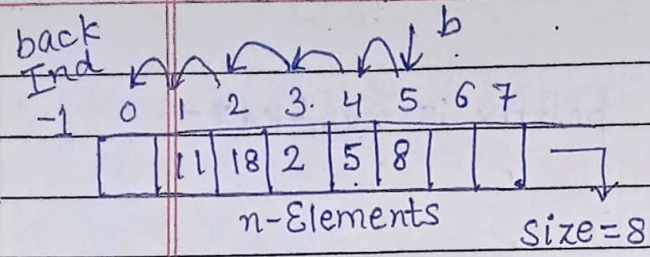
Note:

Queue can be Implemented in various ways!

→ Arrays
→ using Linked list
→ using other ADT's

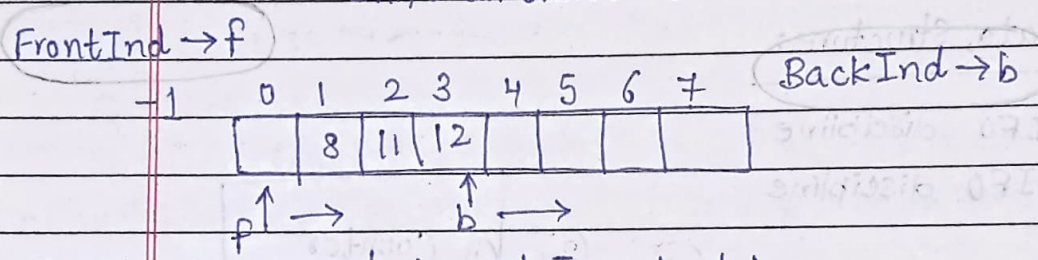
Lec-39 Queue Implementation: Array Implementation of Queue in Data Structure:





Insert: → Increment backInd] O(1)
(Enqueue) → Insert at backInd]

Remove/dequeue: Remove element at Ind 0 | O(n)
Shift all Elements



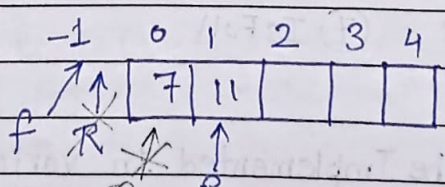
Insert: Increment b and Insert at b
Remove: Increment f and remove element at f.

First Element → frontInd+1.
Rear Element → BackInd
Queue Empty → f=b
Queue Full → b=Size-1

Imp

Lec-40 Array Implementation of Queue and it's Operation in Data Structure:

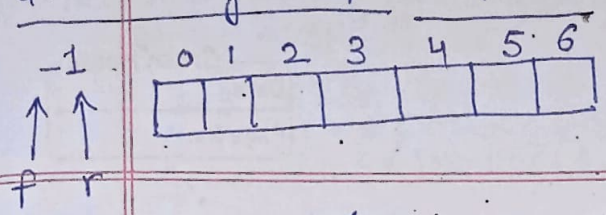
Queue Using Arrays:




```
struct Queue {
    int size;
    int f;
    int r;
    int *arr;
};
```

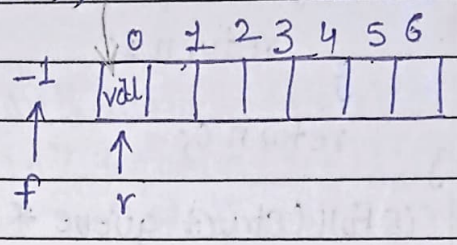
```
int main() {
    struct Queue q;
    q.size=10;
    q.f=q.r=-1;
    q.arr=(int*)malloc(q.size*sizeof(int));
}
```

Queue Using Arrays - Enqueue:




```
struct Queue {
    int size;
    int f;
    int r;
    int *arr;
};
```

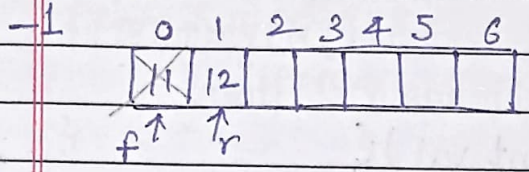
```
void enqueue(struct Queue *q, int val){
    if(isFull(q)){
        printf("Queue Overflow");
    }
    else {
        q->r = q->r+1;
        q->arr[q->r] = val;
    }
}
```



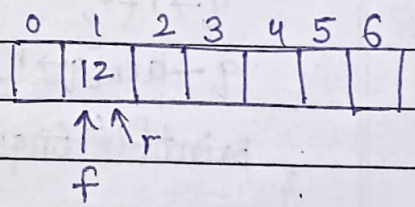
Queue Using Arrays - dequeue:

```
if(q->r == q->size-1)
    return 1;
```

isFull(q)




```
int dequeue(struct Queue *q){
    int a = -1;
    if(q->f == q->r){
        printf("No Element to dequeue");
    }
    else {
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}
```



Lec-41. C Code for Queue and it's Operations Using Arrays in Data Structure.

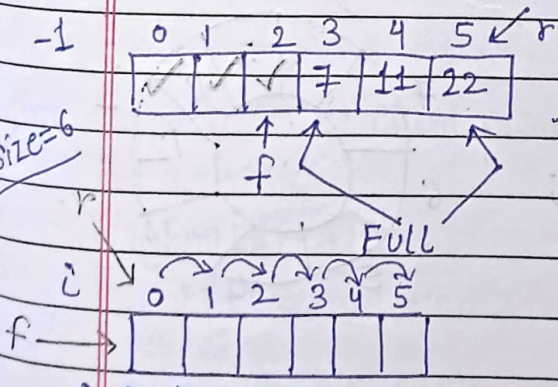
```
#include<stdio.h>
#include<stdlib.h>
```

```
struct queue {
    int size;
    int f;
    int r;
    int* arr;
};

int isEmpty(struct queue *q){
    if(q->r==q->f){
        return 1;
    }
    return 0;
}
int isFull(struct queue *q){
    if(q->r==q->size-1){
        return 1;
    }
    return 0;
}

void enqueue(struct queue * q, int val){
    if(isFull(q)){
        printf("This Queue is full\n");
    }
    else {
        q->r++;
        q->arr[q->r]=val;
        printf("Enqued Element: %d\n", val);
    }
}

int dequeue(struct queue *q){
    int a= -1;
    if(isEmpty(q)){
        printf("This Queue is Empty\n");
    }
    else {
        q->f++;
        a= q->arr[q->f];
    }
    return a;
}
```

```
int main (){
struct queue q;
  q.size = 4;
  q.f = q.r = 0; -1;
  q.arr = (int*) malloc (q.size * sizeof (int));
// Enqueue few Elements
  enqueue (&q, 12);
  enqueue (&q, 15);
  enqueue (&q, 1);
  printf ("Dequeuing Element %d \n", dequeue (&q));
  printf ("Dequeuing Element %d \n", dequeue (&q));
  printf ("Dequeuing Element %d \n", dequeue (&q));
  enqueue (&q, 45);
  enqueue (&q, 36);
  enqueue (&q, 15);
  if (isEmpty (&q)) {
      printf ("Queue is Empty \n");
  }
  if (isFull (&q)) {
      printf ("Queue is Full \n");
  }
  return 0;
}
```

Green House
Date
Page No.

Lec-42 Introduction to Circular Queue in Data Structure:



Drawbacks of Queue using Arrays:
→ space is not used efficiently.

i = i+1; = Linear Increment

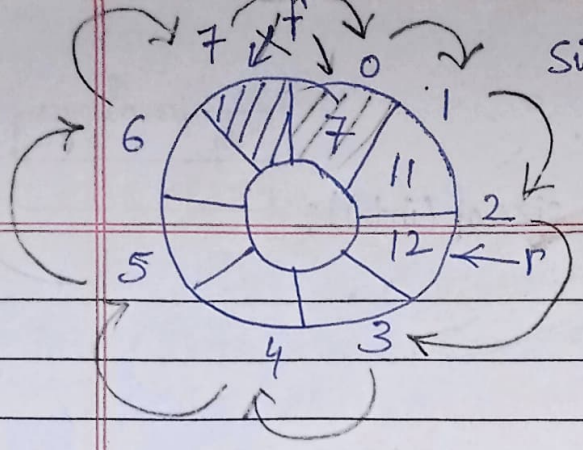
i = (i+1) % size

→ Circular Increment

Circular Increment :

i = (i+1) % size

0 → (0+1) % 6 = 1
1 → (1+1) % 6 = 2
4 → (4+1) % 6 = 5
5 → (5+1) % 6 = 0



Green House

Date

Page No.

Circular Queue.

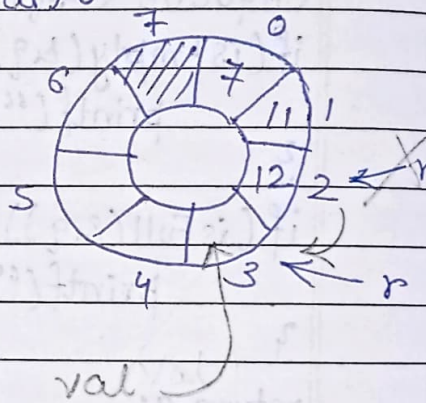
$i = (i+1) \% size$

Lec-43 enqueue(), dequeue() and other operations on Circular Queue.

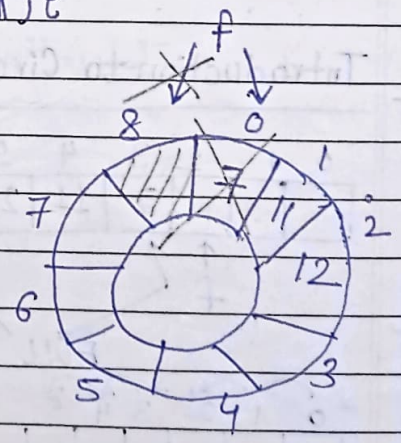
Note: $q.r = q.f = 0$

Coding Circular Queue:

```
void enqueue(struct Queue *q, int val){
    if((q->r+1)%q->size == q->f){
        printf("Queue Overflow");
    }
    else{
        q->r = (q->r+1)%q->size;
        q->arr[q->r] = val;
    }
}
```




```
int dequeue(struct Queue *q, int val){
    int val = -1;
    if(q->f == q->r){
        printf("Empty Queue");
    }
    else{
        q->f = (q->f+1)%q->size;
        val = q->arr[q->f];
    }
    return val;
}
```



Lec-44 C Code for Circular Queue & Operations on Circular Queue:

```
#include<stdio.h>
#include<stdlib.h>
struct CircularQueue{
    int size;
    int f;
    int r;
    int *arr;
};
int isEmpty(struct CircularQueue *q){
    if(q->f==q->r){
        return 1;
    }
    else{
        return 0;
    }
}
int isFull(struct CircularQueue *q){
    if((q->r+1)%q->size==q->f){
        return 1;
    }
    else{
        return 0;
    }
}
void enqueue(struct CircularQueue *q, int val){
    if(isFull(q)){
        printf("Queue Overflow");
    }
    else{
        q->r=(q->r+1)%q->size;
        q->arr[q->r]=val;
        printf("Enqued Element: %d \n",val);
    }
}
int dequeue(struct CircularQueue *q){
    int a=-1;
    if(isEmpty(q)){
        printf("Queue underflow");
    }
    else{
        q->f=(q->f+1)%q->size;
        a=q->arr[q->f];
    }
    return a;
}
```

q.f=q.r=0

```
int main(){
    struct circularQueue q;
    q.size=4;
    q->f=q.r=0;
    q.arr=(int*)malloc(q.size*
                sizeof(int));
    // Enqueue few elements
    enqueue(&q,12);
    enqueue(&q,15);
    enqueue(&q,1);
    printf("dequeuing Element: %d
        \n",dequeue(&q));
    enqueue(&q,45);
    if(isEmpty(&q)){
        printf("Queue is Empty\n");
    }
    if(isFull(&q)){
        printf("Queue is Full\n");
    }
    return 0;
}
```

```
Enqued Element: 12
Enqued Element: 15
Enqued Element: 1
dequeuing Element: 12
Enqued Element: 45
Queue is Full
```

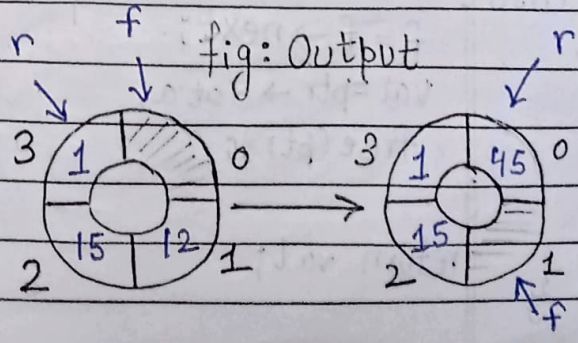
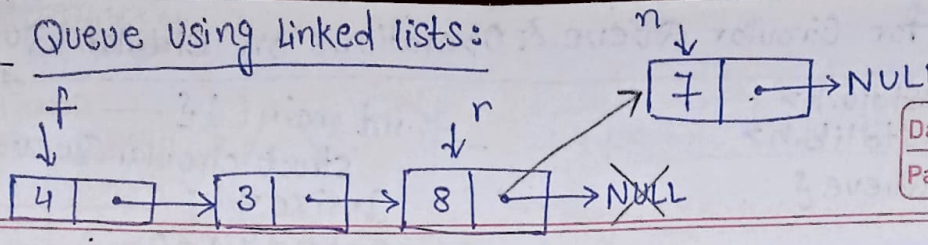


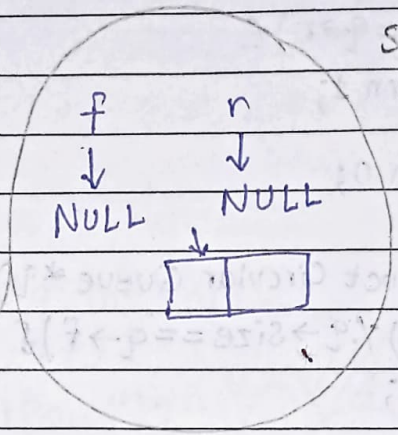
fig: Output

Lec-45 Queue using linked lists:



Green House
Date
Page No.

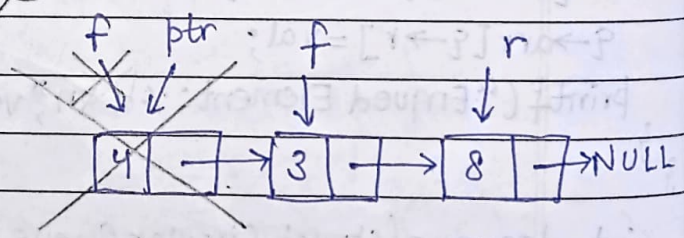
```
void enqueue (struct Node* f, struct Node* r, int val)
{
  struct Node* n = (struct Node*) malloc (sizeof (struct Node));
  if (n == NULL) {
      printf("Queue Full");
  }
  else {
      n->data = val;
      n->next = NULL;
      if (f == NULL) {
          f = r = n;
      }
      else {
          r->next = n;
          r = n;
      }
  }
```



Note

[A] Condition for Queue Empty
f == NULL

[B] Condition for Queue Full
n == NULL




```
int dequeue (struct Node *f) {
    int val = -1;
    struct Node* ptr = f;
    if (f == NULL) {
        printf("Queue Empty");
    }
    else {
        f = f->next;
        val = ptr->data;
        free(ptr);
    }
    return val;
}
```

When Queue is Not Empty

Lec-46 Implementing Queue using Linked List in C Language:

Green House
Date
Page No.

```
#include<stdio.h>
#include<stdlib.h>
struct Node *f = NULL;
struct Node *r = NULL;
```

Globaly Declared

```
struct Node {
    int data;
    struct Node * next;
};

void linkedlistTraversal(struct Node *ptr) {
printf("Printing the Elements of this Linked list \n");
while (ptr != NULL)
{
 printf("Element: %d \n", ptr->data.);
  ptr=ptr->next;
}
}

void enqueue(int val)
 {
  struct Node* n= (struct Node*) malloc (sizeof (struct Node));
  if (n== NULL) {
      printf("Queue is Full");
  }
else {
    n->data=val;
    n->next=NULL;
  if(f==NULL) {
       f=r=n;
  }
else {
    r->next=n;
     r=n;
}
}
}

int dequeue () {
   int val= -1;
  struct Node *ptr = f;
```

```
    if(f==NULL){
        printf("Queue is Empty\n");
    }
else {
    f=f->next;
    val=ptr->data;
    free(ptr);
}
return val;
}

int main(){
    linkedListTraversal(f);
    printf("Dequeuing Element: %d\n", dequeue());
    enqueue(34);
    enqueue(4);
    enqueue(7);
    enqueue(17);
    printf("Dequeuing Element: %d\n", dequeue());
    printf("Dequeuing Element: %d\n", dequeue());
    printf("Dequeuing Element: %d\n", dequeue());
    printf("Dequeuing Element: %d\n", dequeue());
    linkedListTraversal(f);

    return 0;
}
```

Green House

Date

Page No.

Output

```
Printing the Elements of this linked list
Dequeuing Queue is Empty
Dequeuing Element: -1
Dequeuing Element: 34
Dequeuing Element: 4
Dequeuing Element: 7
Dequeuing Element: 17
Printing the elements of this linked list.
```

Lec-47. Double-Ended Queue in Data Structure (DE-Queue Explained):

[Queue]
→ Insertion from Rear
→ Deletion from Front
→ FIFO

[DE-Queue]
→ Insertion from Rear + Front
→ Deletion from Rear + Front
→ FIFO

→ Two Types of DE-Queue.
1. Restricted Input DE-Queue
2. Restricted Output DE-Queue

Restricted Input DE-Queue: Insertion from front Not Allowed

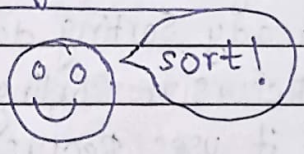
Restricted Output DE-Queue: Deletion from rear is not Allowed

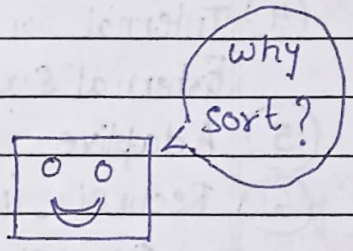
DE-Queue ADT:

1. Data → Same as Queue
2. Operations → isEmpty()
isFull()
Initialize()
print()
enqueueF()
enqueueR()
dequeueF()
dequeueR()

To do
To do

Lec-48 Introduction to Sorting Algorithms:



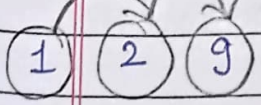


Two ways of Sorting
1) Ascending Order → 1, 2, 7, 8, 9
2) Descending Order → 9, 8, 7, 2, 1

Sort by: Newest!
Top Rated!

Swiggy/Zomato → Restaurants → Sort by Rating → Algo ✓
→ Sort by Price → Algo ✓

Linear Binary



20 71 33 → yes, 9 is present

✓ → ↓ n elements

Lec-47. Double-Ended Queue in Data Structure (DE-Queue Explained):

[Queue]
→ Insertion from Rear
→ Deletion from Front
→ FIFO

[DE-Queue]
→ Insertion from Rear + Front
→ ~~Insertion~~ Deletion from Rear + Front
→ FIFO
→ Two Types of DE-Queue
1. Restricted Input DE-Queue
2. Restricted Output DE-Queue

Insertion from front Not Allowed

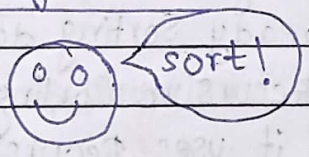
Deletion from rear is not Allowed

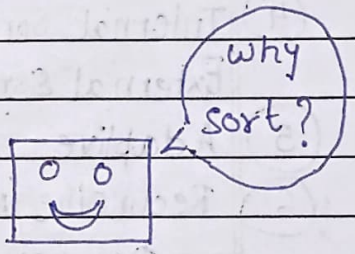
DE-Queue ADT:

1. Data → Same as Queue
2. Operations → isEmpty()
isFull()
Initialize()
print()
enqueueF() To do
enqueueR()
dequeueF()
dequeueR() To do

Lec-48 Introduction to Sorting Algorithms:

1 9 8 2 7 .....



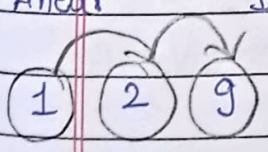


Two ways of Sorting
1) Ascending order → 1, 2, 7, 8, 9
2) Descending order → 9, 8, 7, 2, 1

Sort by: Newest!
Top Rated!

Swiggy/Zomato → Restaurants
→ Sort by Rating → Algo ✓
→ Sort by Price → Algo ✓

Linear Binary



20 71 33 → yes; 9 is present

✓ → ↓ n elements