Circular Queue – In a linear queue, elements were added from rear position & deleted from front. After adding the elements, the rear incremented & finally reached the position SIZE-1 & we got the error message “Queue is Full”.

Similarly after deleting the element, the front incremented. Finally the front crossed the rear which was @ position SIZE-1 & we got the error message “Queue is Empty”.

However when attempts were made to add elements to this empty queue, we got the message   
“Queue is Full”. Now both the statements are contracdicting each other. How can a queue be full & empty at the same time?

So this problem can be overcome by a concept called as “Circular Queue”.

#define SIZE 5

4

3

2

1

0

// structure of queue is same as linear queue

10,20,30,40,50,60 ~~r~~

void addq(struct queue \*pq, int no) ~~r~~ 30

{ f 20

~~pq->rear++;~~

pq->rear = (pq->rear+1)%SIZE;

pq->data[pq->rear] = no;

if (pq->front==-1) ~~f~~,~~r~~ ~~10~~ 40 ~~r~~

pq->front = 0; r(2) 60

}

f,r = -1 50

r

Queue is full only when we have front 5

ahead of rear on a circular path

int isqfull(struct queue \*pq)

{

if (pq->front == (pq->rear+1)%SIZE)

return 1;

return 0;

} f f

~~20~~ ~~30~~

2

1

From the diagram, it appears that the queue is

empty when the front is just ahead of rear.

3

However this condition is used for qfull also. ~~40~~ ~~f~~

0

This will again lead to contradiction. ~~f~~  ~~10~~

4

Solution: f(2)

During insertion, the front is stationary & rear ~~50~~

is moving. Here if it sees front ahead of it,

we say that the queue is full. r, ~~f~~

During deletion, rear is stationary & front is moving.

At some point, the front crosses rear & then the queue

becomes empty. So the moment front crosses rear,

we will reset front & rear to -1 which is a different

condition for empty queue.