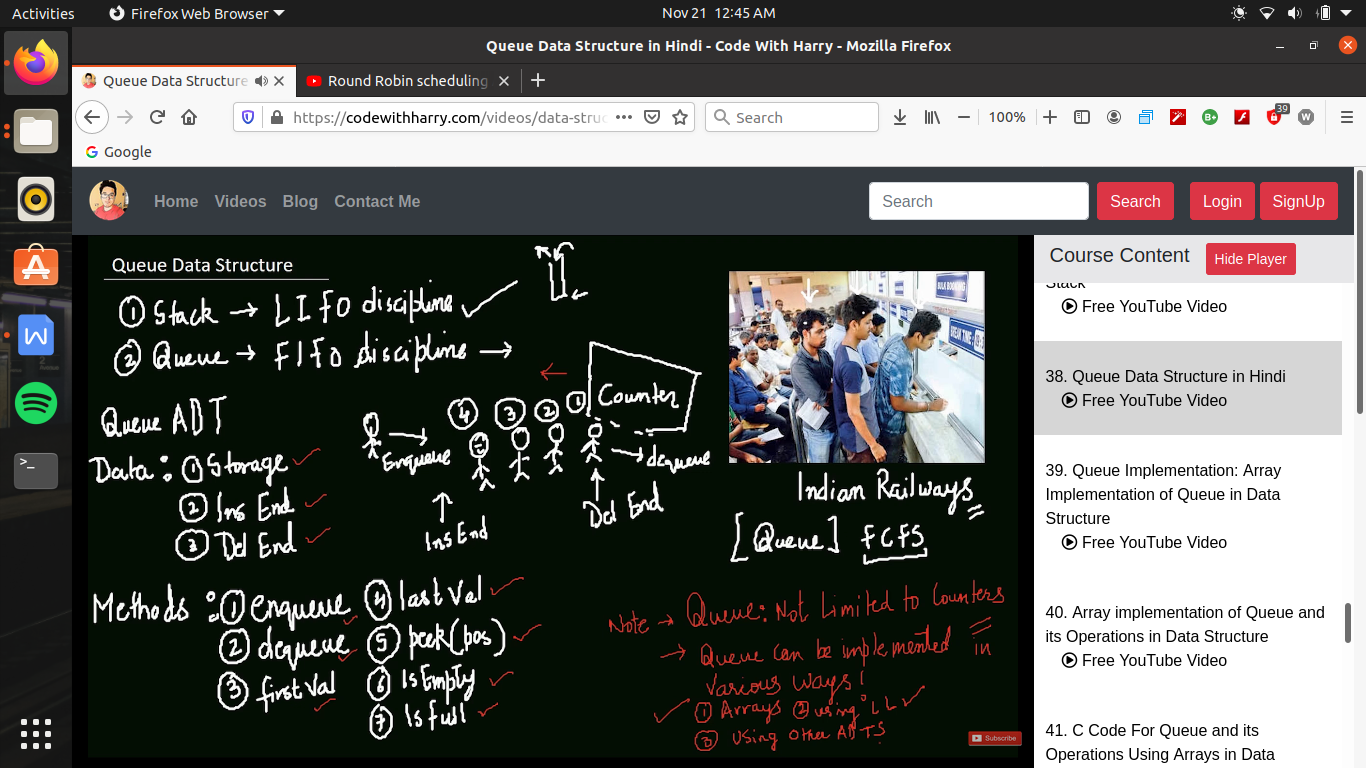
queue

stack ->LIFO (last in first out )

queue ->FIFO (first in first out)



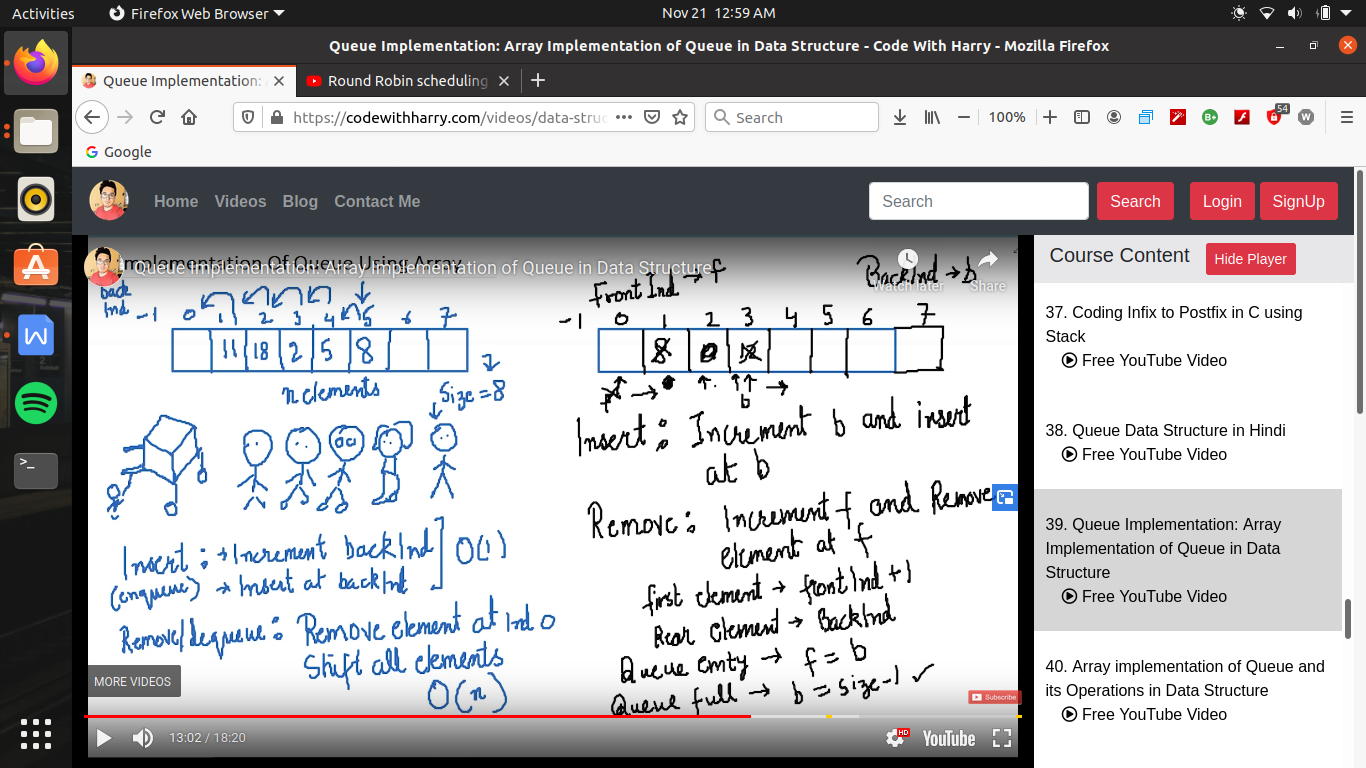
queue using array :-

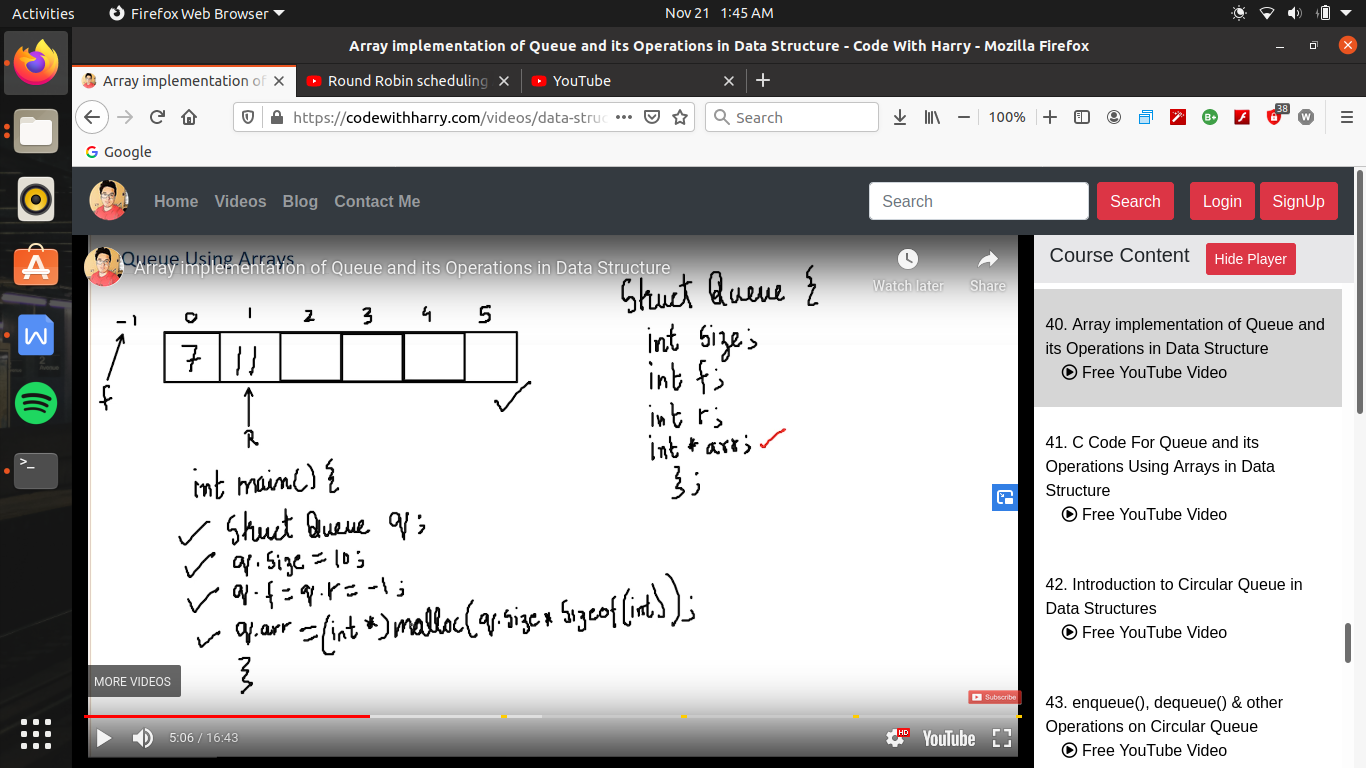
in queue using array if we have to remove(dequeue) so after removal we have to swift all elements one by one

toh es problem ko sovle karane ke leye hum 2 index lete hai ex-> fornt\_index and back\_index(es me pahle front end ko age bada dege fir remove karege )

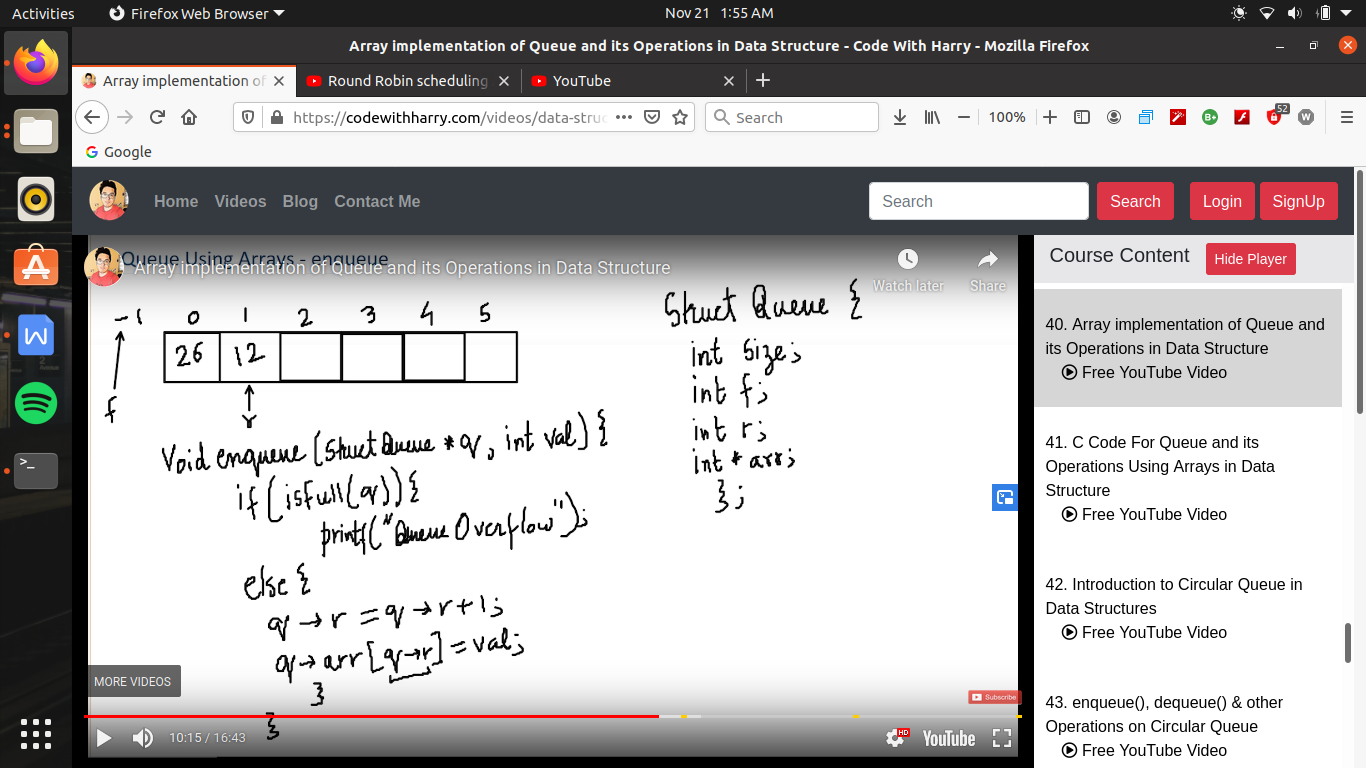
and id front index =back index then it is a queue empty condition

queue full condition when back\_index = size -1;





enqueue operation ->



Drawbacks of queue using array :-

1. space is not used efficiently (as because array me already space allocated hai so jaise jaise front\_index age badata jayega waise waise age wali index khali hoti jayegi , but we can not use this space again)

so to resolve this issue we again set the back\_index on the front

to this we will use circular increment to do that

here

i= i+1; //this is linear increment

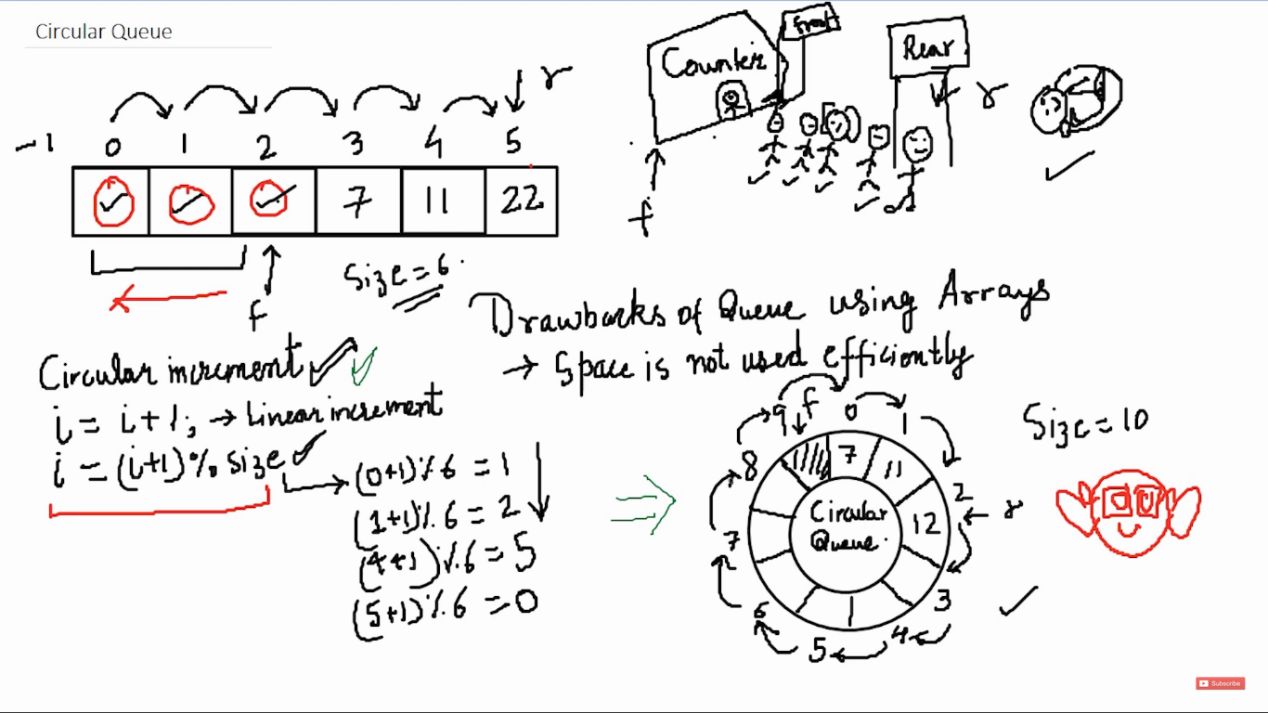
now for circular increment we use modulus operator i.e(%)

by i=(i+1)%size

circular queue -

we made circular queue becaue we don’t want ot waste your elements

but es me bhi ek front wali space use nhi kr pa rahe hai (i.e pahali wali element )



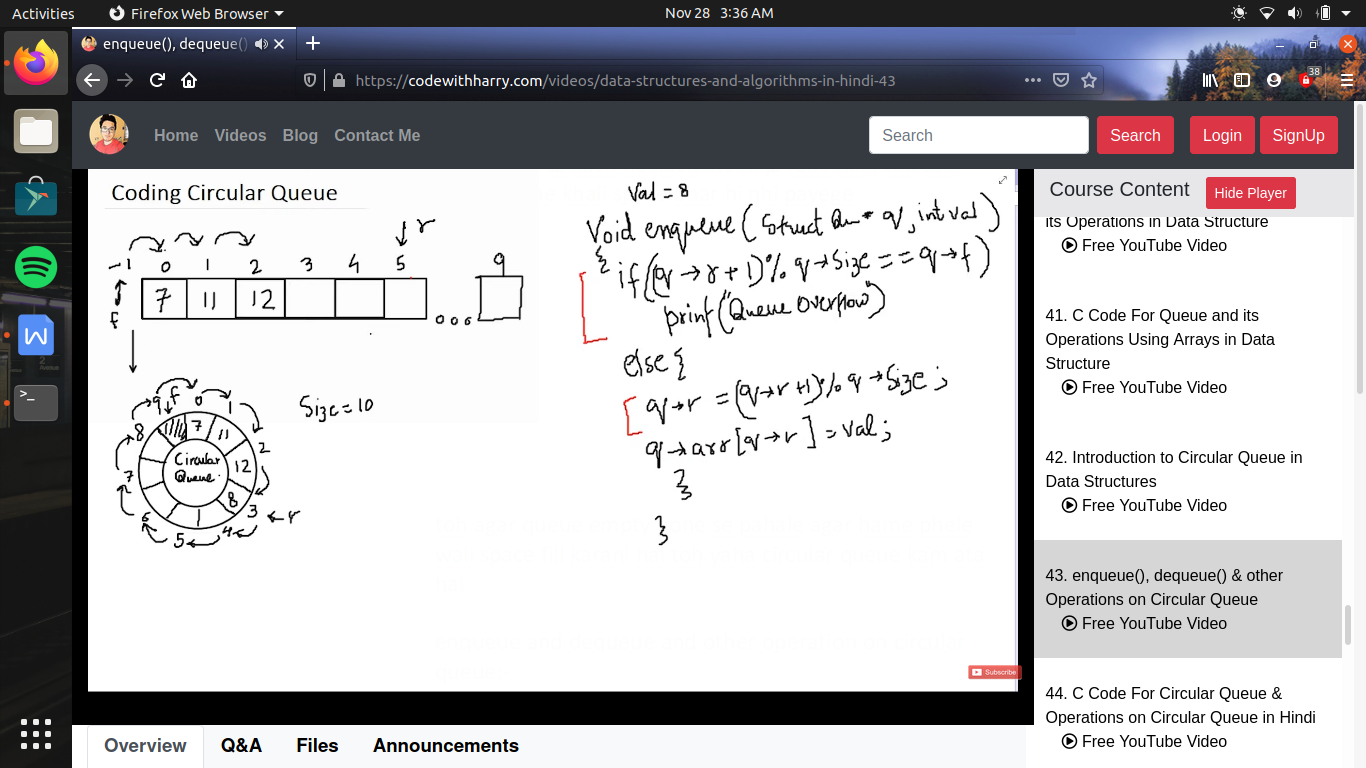
toh es ko emplement karane ke leye hum yeh kh skate hai ki agra front = rear ho jata hai toh hum front ko wapas start me dal dege

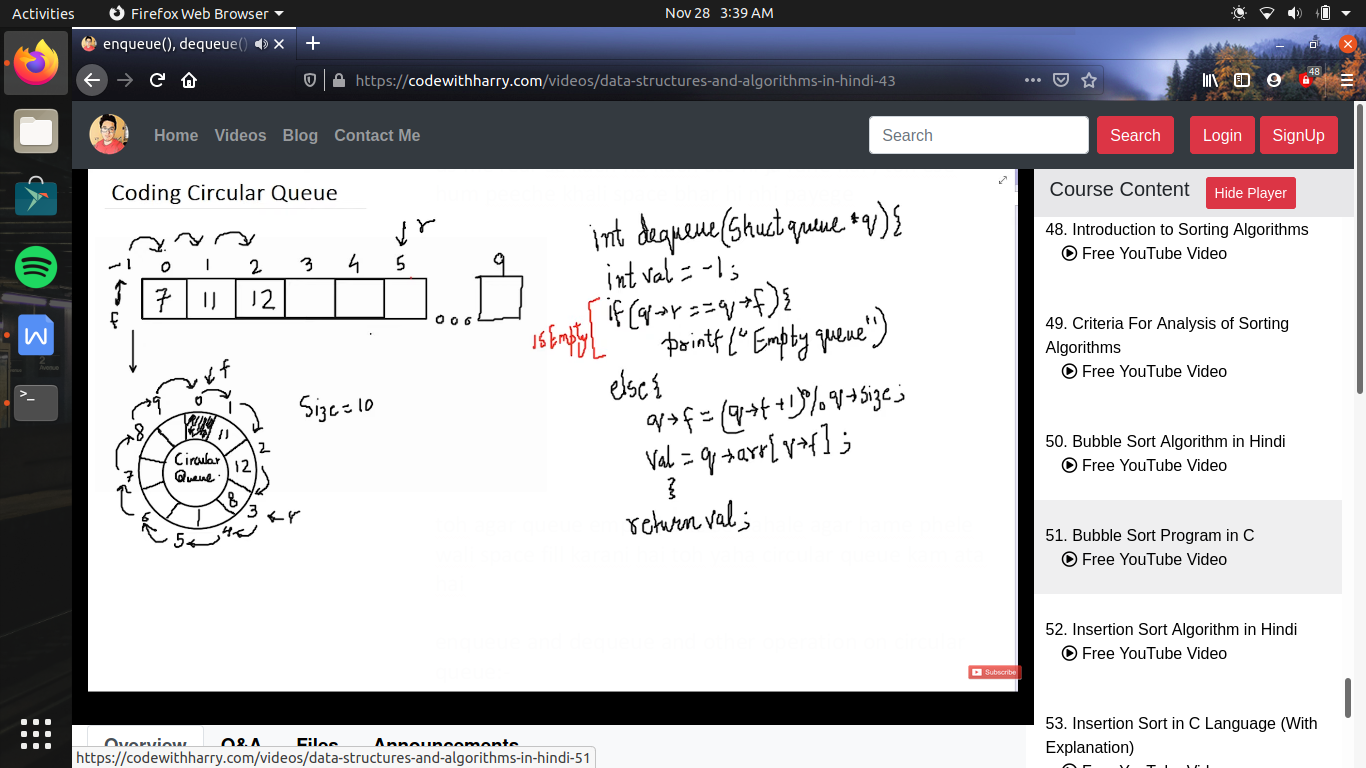
but ese jab tak queue emply nhi hoti tab tak khali space fill nhi kr payege

or yeh bhi ho sakata hai ki queue empty ho hi na kabhi us me rear se kuch na kuch ate hi ja rahe hai , toh ese hum peeche khali space bhar hi nhi payege

toh agar queue empty hone se pahale agar hame phele wali space fill karani hai toh yaha circular queue kam ata hai

enqueue and dequeue and other operation on circular queue:-





#include<stdio.h>

#include<stdlib.h>

struct circular\_queue

{

int size;

int f;

int r;

int \* arr;

};

int is\_empty (struct circular\_queue \* q)

{ if (q->f==q->r)

return 1;

else

return 0;

}

int is\_full (struct circular\_queue \*q)

{

if ((q->r+1)%q->size == q->f)

return 1;

else

return 0;

}

void enqueue (struct circular\_queue \*q)

{

if (is\_full(q))

{printf("queue overflow \n");}

else

{ q->r=(q->r+1)%q->size;

printf("enter the value of element to be stored \n");

scanf("%d",&q->arr[q->r]);

}

}

int dequeue(struct circular\_queue \*q)

{

if(is\_empty(q))

printf("queue is empty \n");

else

q->f=(q->f+1)%q->size;

int a =q->arr[q->f];

return a;

}

void display(struct circular\_queue \*q)

{

int i=q->f;

while(i != q->r)

{

printf("%d \t",q->arr[i]);

i=(i+1)%q->size;

}

printf("%d \n",q->arr[q->r]);

}

void choice(struct circular\_queue \*q)

{

while (1)

{

int a;

printf("please select your choice \n");

printf("1. enqueue \n2. dequeue\n3. display\n4. exit\n");

scanf("%d",&a);

switch(a)

{ case 1: enqueue(q);

break;

case 2: dequeue(q);

break;

case 3: display(q);

break;

case 4: exit(0);

default:

printf("invalid input please try again \n\n\n");

}

}

}

int main()

{

int s;

printf("please enter the size of queue \n");

scanf("%d",&s);

struct circular\_queue cq;

cq.size=s;

cq.f=cq.r=0; // yaha pe hum front and rear index ko zero hi rakhate hai kyu ki jab bhi hum is\_full check karege toh abhi bhi q->f =-1; nhi ho payega agar ek bar start ho gaya ek es bad

cq.arr=(int\* )malloc(cq.size\*sizeof(int));

choice(&cq);

return 0;

}

