❖ INTRODUCTION TO QUEUE (FIFO)

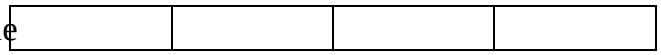
1. Queue

A queue is an ordered collection of items from which items may be deleted at one end (collect the front of the queue) and into which items may be inserted at the other end (called the nearer of the queue).

2. Initialize Queue

The first element inserted into the queue is the first element to be removed. For this reason a queue is sometimes called a **FIFO** (first-in first-out) list as opposed to the stack, which is a LIFO (last-in first-out).

Initial queue



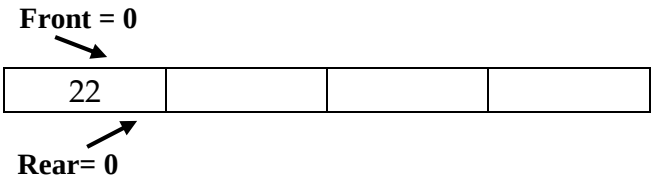
Front = -1
Rear= -1

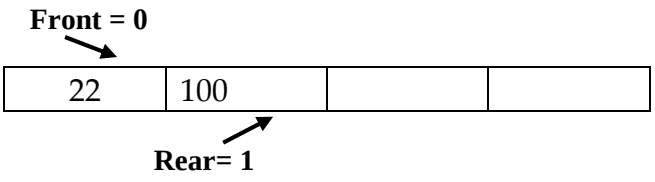
3. Queue Operations

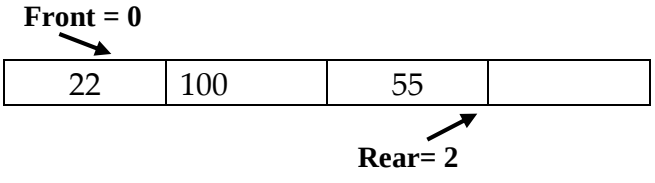
The two primary queue operations are **enqueuing** and **dequeuing.**

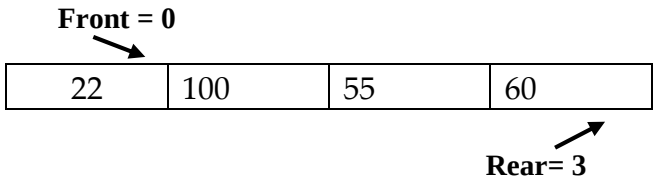
- To enqueue means to insert an element at the rear of a queue.

Enqueuing operations



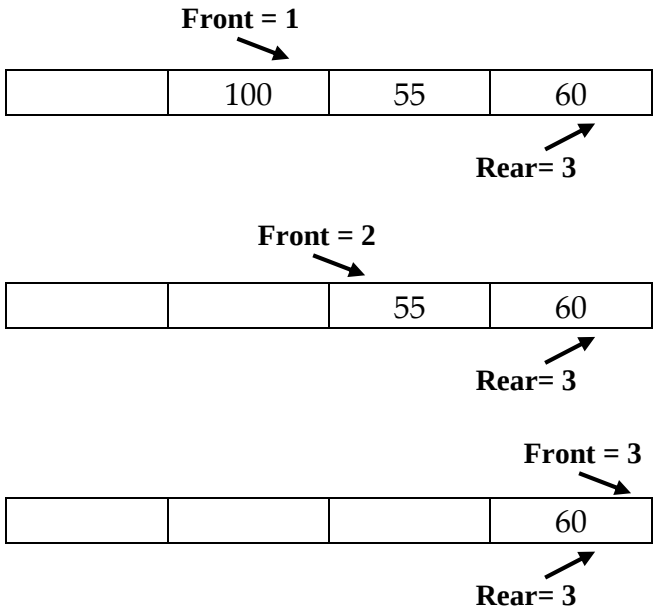




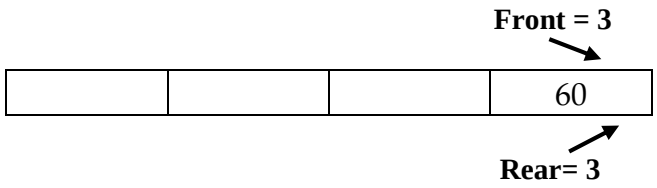


- To dequeue means to remove an element from the front of a queue.

Dequeuing operations

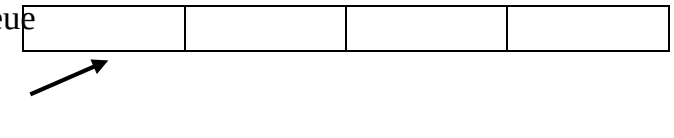


- An empty queue can be signified by setting both front and rear indices to –1.



After dequeuing

An empty queue



4. Queue Algorithms

There are two algorithms to insert (**Enqueuing**) and remove (**Dequeuing**) items into and from Queue.

- Insert Algorithm (**Enqueuing**)

1. [overflow]
   if R>=N-1
   Then Over flow
2. [increment Rear]
   Rear ← Rear+1
3. [insert element]
   Queue[Rear] ← New element
4. [set Front]
   If Front=-1
   Then Front =0

- Delete Algorithm (**Dequeuing**)

1. [underflow]
   if Front= -1
   Then under flow
2. [Delete element]
   Element ← Queue[Front]
3. [check empty Queue]
   If Front =Rear
   then Front =Rear= -1
   Else Front ← Front+1

5. Queue Applications

- A waiting line is a good real-life example of a queue.(A waiting line in a store, at a service counter)
- Equal-priority processes waiting to run on a processor in a computer system

6. Queue Problem: False-Overflow Issue First

- For Queue array size [50] .Suppose 50 calls to enqueue have been made, so now the queue array is full
- Assume 4 calls to dequeue( ) are made
- Assume a call to enqueue( ) is made now. The tail part seems to have no space, but the front has 4 unused spaces; if never used, they are wasted.

Solution: A Circular Queue

// RUN OPERATION OF QUEUE

```
        #include<iostream.h>
struct queue
{
int rear;
int front;
int   item [2] ;
};
main()
{
queue q1;
q1.front=q1.rear=-1;
char c;  int item;
// push to stack
cout<<" \n Do  you want to enqueuing in to queue (y,n): ";
cin>>c;
while(c=='y')
{
if(q1.rear>=2)
    {cout<<"\n error...the queu is full"<<endl; break;}
        else
         { cout<<"\n enter item: ";
         cin>>item;
```

```
            q1.rear ++;
            q1.item [q1.rear]=item;
            if (q1.front =-1) {q1.front=0;}
           }
cout<<" \n Do  you want to  enqueuing in to queue(y,n): ";
          cin>>c;
}
//pop from stack
 cout<<" \n Do  you want to dequeuing from queue (y,n): ";
 cin >>c;
 while (c=='y')
 {        //cout<<"front="<<q1.front;
 if(q1.front <0)
                  { cout<<"\n error…the stack is empty \n"<<endl;break; }
  else
                   {
                   item=q1.item[q1.front];
                   q1.front++;
                    cout<<item<<"  :is dequeued\n ";
                    if (q1.front>q1.rear)
                    {q1.front=q1.rear=-1; break;}// returen to initial value
                   }

         cout<<" \n Do  you want to dequeuing from queue (y,n): ";
 cin >>c;
 }
 }
```