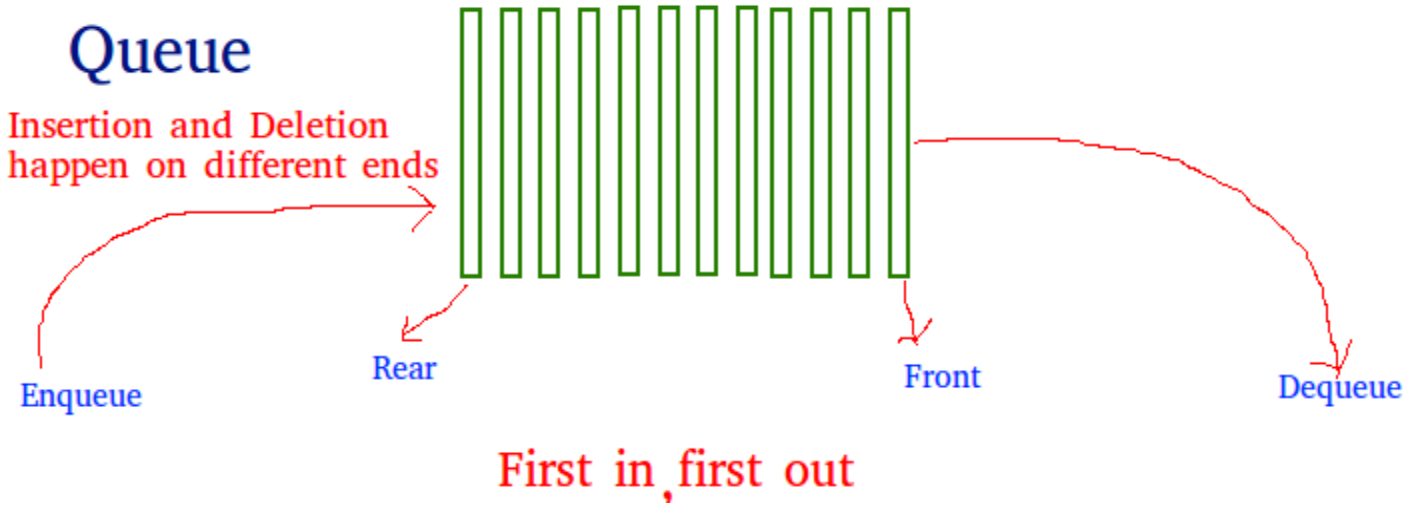
Data Structure and Algorithm

Presented by: Er. Aruna Chhatkuli

Nepal College of Information Technology,
Balkumari, Lalitpur

Queue

- A queue is logically a first in first out (FIFO or first come first serve) linear data structure.
- A queue is an ordered collections of items from which items may **be deleted at one end called the front** of the queue and in to which items may be **inserted at the other end called rear** of the queue.



Queue

- The basic operations (Primitive Operations) also called as Queue ADT that can be performed on queue are:
  1. **Enqueue() or Insert() (or add)** an element to the **queue (push)** from the rear end.
  2. **Dequeue() or Delete() (or remove)** an element from a **queue (pop)** from front end.
  3. **Front:** return the object that is at the front of the queue without removing it.
  4. **Empty:** return true if the queue is empty otherwise return false.
  5. **Size:** returns the number of items in the queue

TYPES OF QUEUE

i. **Linear Queue** or Simple queue

ii. **Circular queue**

iii. **Double ended queue (de-queue)**

iv. **Priority queue:** Priority queue is generally implemented using linked list

Enqueue

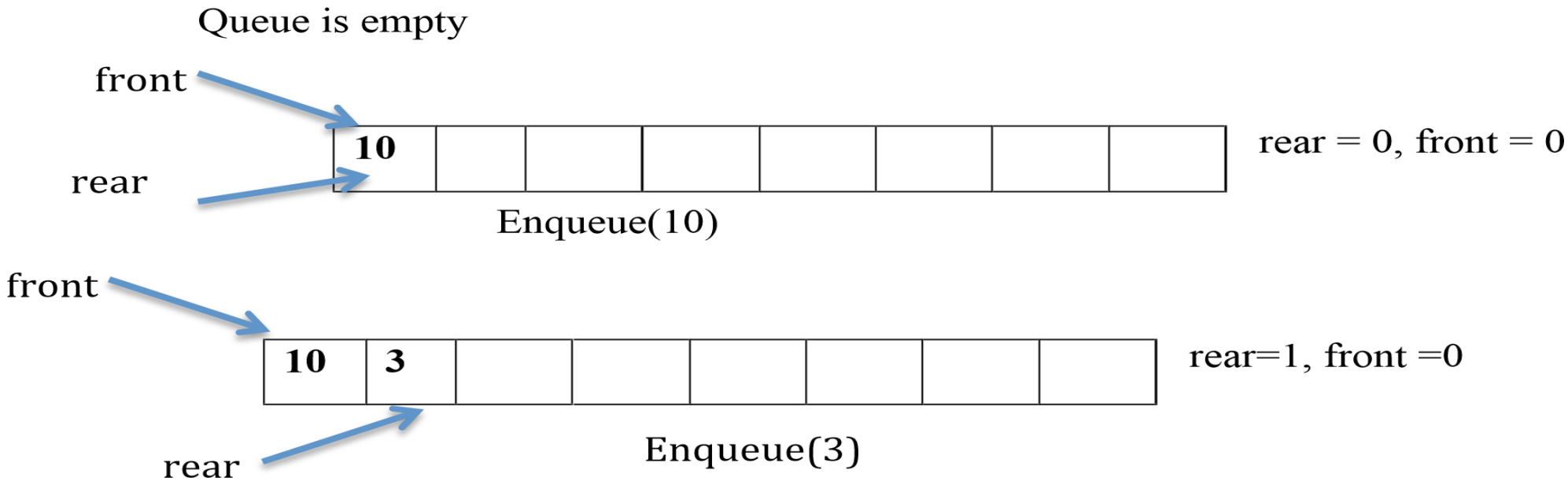
- Enqueue operation will insert (or add) an element to queue, at the rear end, by incrementing the array index.
- Pop operation will delete (or remove) from the front end by decrementing the array index and will assign the deleted value to a variable.
- Initially rear is set to -1 and front is set to -1.
- The queue is empty whenever rear < front or both the rear and front is equal to -1.
- Total number of elements in the queue at any time is equal to rear-front+1, when implemented using arrays. Below are the few operations in queue.

Enqueue

❖ Below are the few operations in queue:

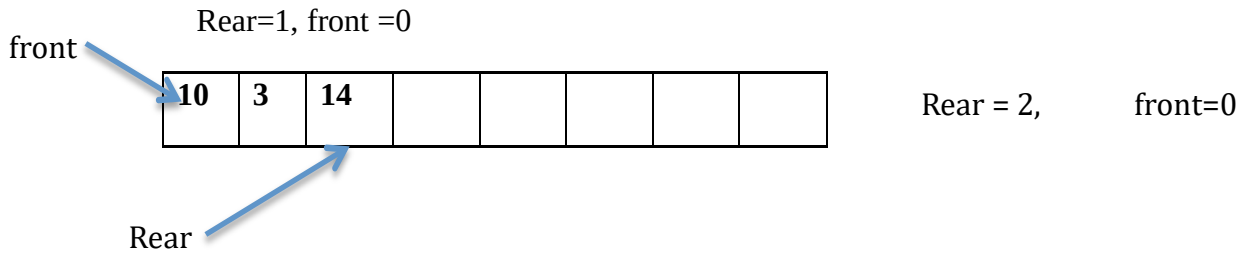
Queue is an empty.

Front=-1, rear=-1

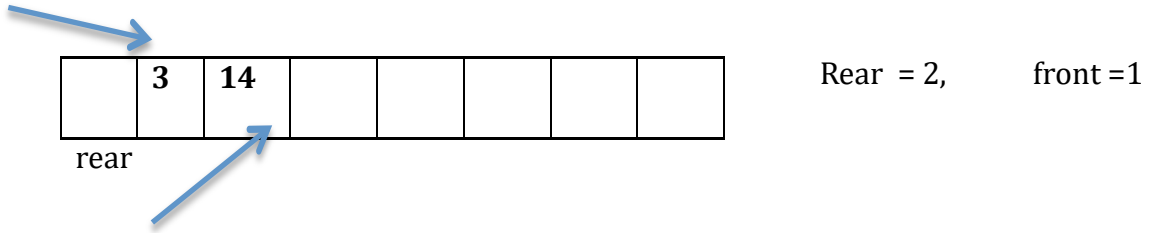


Enqueue

❖ Below are the few operations in queue:



Enqueue(14)



❖ x = Dequeue() (i.e. x = 10)

Note: During the insertion of first element in the queue we always increment the front by one.

Enqueue

Queue can be implemented in two ways:

1. **Using arrays (static)**
2. **Using pointers (dynamic)**

- If we try to dequeue (or delete or remove) an element from queue when it is empty, underflow occurs. It is not possible to delete (or take out) any element when there is no element in the queue.
- Suppose maximum size of the queue (when it is implemented using arrays) is 50. If we try to Enqueue (or insert or add) an element to queue, overflow occurs.
- When queue is full it is naturally not possible to insert any more elements.

Algorithm for insertion

- **Step 1:** IF REAR = MAX – 1

  Write OVERFLOW
  Go to step
  [END OF IF]

- **Step 2:** IF FRONT = -1 and REAR = -1

  SET FRONT = REAR = 0
  ELSE
  SET REAR = REAR + 1
  [END OF IF]

- **Step 3**: Set QUEUE[REAR] = NUM
- **Step 4**: EXIT

Deleting An Element From Queue:

1. If (rear< front) or if(front ==-1 && rear == -1); [checking for queue is empty]
   i. front = -1, rear = –1;(optional)
   ii. Display “The queue is empty”
   iii. Exit
2. Else
   i. Data = Q [front]
   ii. Front = front +1
3. Exit

Advantages of Queue

- **A large amount of data can be managed efficiently with ease.**
- **Operations such as insertion and deletion can be performed with ease as it follows the first in first out rule.**
- **Queues are useful when a particular service is used by multiple consumers.**
- **Queues are fast in speed for data inter-process communication.**
- **Queues can be used in the implementation of other data structures.**

Disadvantages of Queue

- The operations such as insertion and deletion of elements from the middle are time consuming.
- Limited Space.
- In a classical queue, a new element can only be inserted when the existing elements are deleted from the queue.
- Searching an element takes O(N) time.
- Maximum size of a queue must be defined prior.

Application of Queue

- **Multi programming:** Multi programming means when multiple programs are running in the main memory. It is essential to organize these multiple programs and these multiple programs are organized as queues.
- **Network:** In a network, a queue is used in devices such as a router or a switch. another application of a queue is a mail queue which is a directory that stores data and controls files for mail messages.

Application of Queue

- **Job Scheduling:** The computer has a task to execute a particular number of jobs that are scheduled to be executed one after another. These jobs are assigned to the processor one by one which is organized using a queue.
- **Shared resources:** Queues are used as waiting lists for a single shared resource.

Real-time application of Queue:

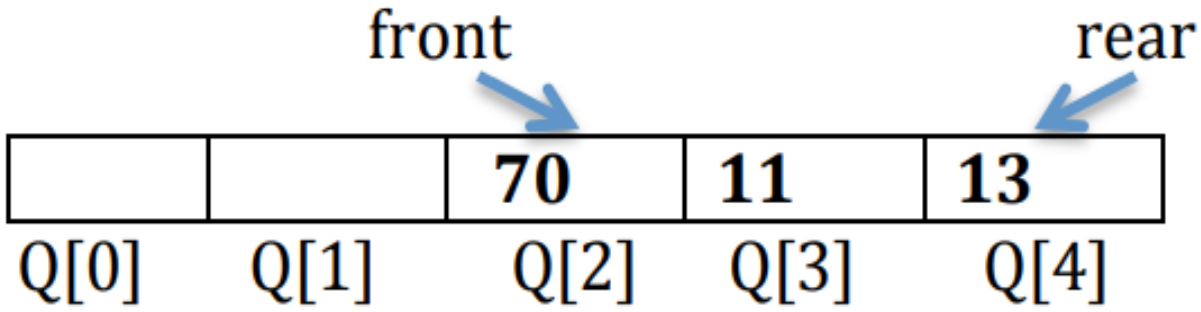
- ATM Booth Line
- Ticket Counter Line
- Key press sequence on the keyboard
- CPU task scheduling
- Waiting time of each customer at call centers.

Circular Queue

- Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.

Circular Queue

Suppose a queue Q has maximum size 5, say 5 elements pushed and 2 elements popped.



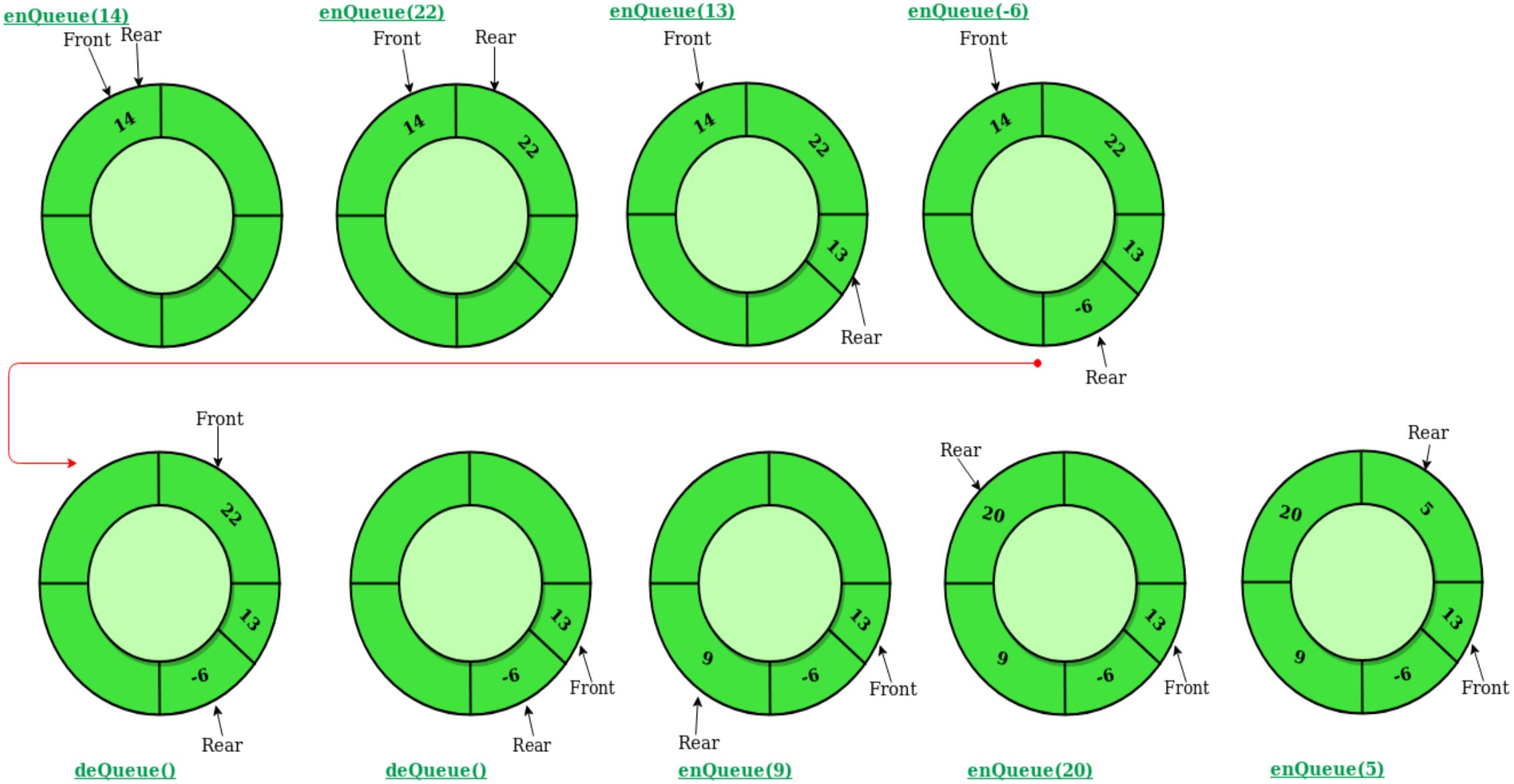
Circular Queue

- Now if we attempt to add more elements, even though 2 queue cells are free, the elements cannot be pushed.
- Because in a queue, elements are always inserted at the rear end and hence rear points to last location of the queue array Q[4]. That is queue is full (overflow condition) though it is empty.
- This limitation can be overcome if we use circular queue.
- In circular queues the elements Q[0],Q[1],Q[2] .... Q[n – 1] is represented in a circular fashion with Q[1] following Q[n].

Circular Queue

- A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.
- Suppose Q is a queue array of 6 elements.
- Enqueue() and Dequeue() operation can be performed on circular.

Circular Queue



Circular Queue

The following figures will illustrate the same:

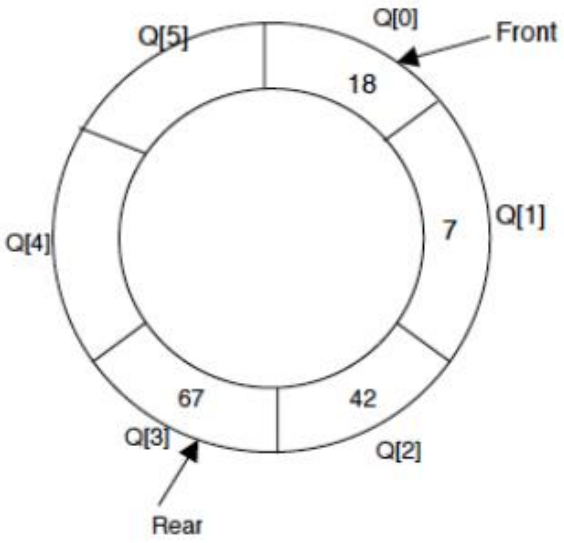


Fig1: A Circular Queue After inserting 18,7,42,67

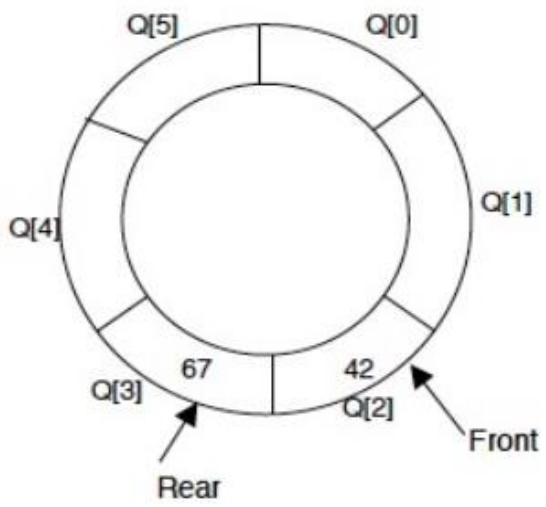


Fig2: Circular Queue after popping 18,7.

Circular Queue:

- After inserting an element at last location Q[5], the next element will be inserted at the very first location (i.e., Q[0]) that is circular queue is one in which the first element comes just after the last element.

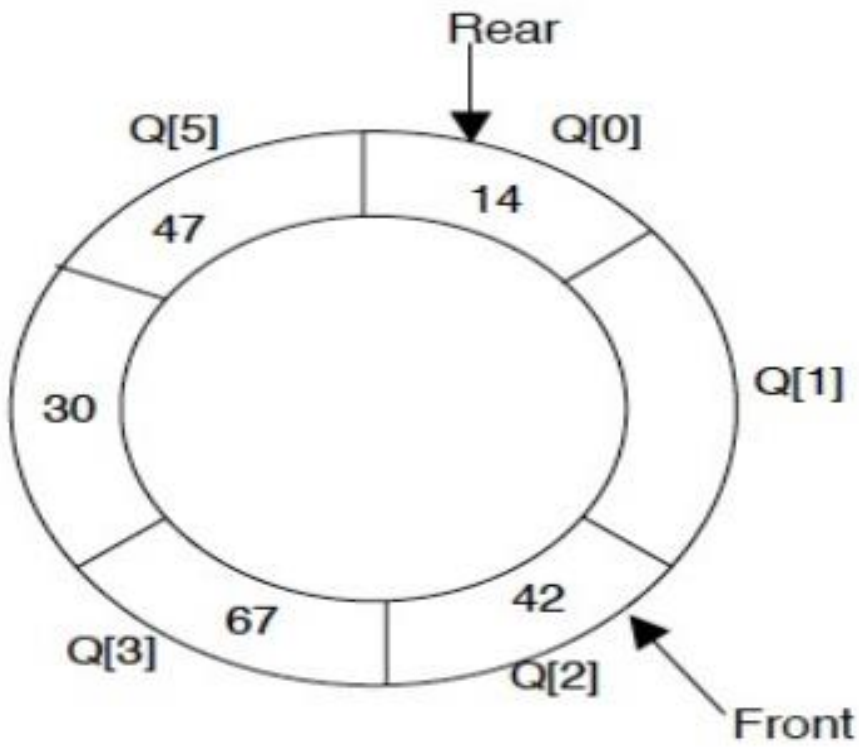


Fig3: Circular Queue after pushing 30, 47, 14

Circular Queue:

- At any time the relation will calculate the position of the element to be inserted Rear = (Rear + 1) % SIZE
- After deleting an element from circular queue the position of the front end is calculated by the relation
  - Front= (Front + 1) % SIZE
- After locating the position of the new element to be inserted, rear, compare it with front. If (rear = front), the queue has only one element.

ALGORITHMS

Let Q be the arrays of some specified size say SIZE.

- FRONT and REAR are two pointers where the elements are deleted and inserted at two ends of the circular queue. DATA is the element to be inserted.

Inserting an element to circular Queue

1. If ((FRONT is equal to 0 && rear = MAX-1) OR front = rear+1) //check for queue full
   i. Display “Queue is full”
   ii. Exit
2. If (FRONT is equal to – 1 && rear == -1) //Empty Queue (inserting first time)
   i. FRONT = REAR = 0
3. Else
   i. REAR = (REAR + 1) % SIZE
4. Input the value to be inserted and assign to variable “DATA”
5. Q [REAR] = DATA
6. Repeat steps 2 to 7 if we want to insert more elements
7. Exit

Deleting an element from a circular queue

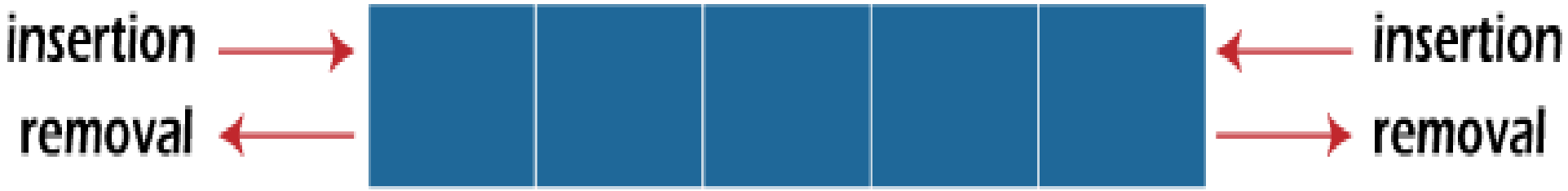
1. If (FRONT is equal to – 1 and rear == -1)
   i. Display “Queue is empty”
   ii. Exit
2. Else
   i. DATA = Q [FRONT]
3. If (REAR is equal to FRONT) //Queue has only one element
   i. FRONT = REAE = –1
4. Else
   i. FRONT = (FRONT +1) % SIZE
5. Repeat the steps 1 to 4, if we want to delete more elements
6. Exit

DEQUES (Double ended queue)

- The deque stands for Double Ended Queue.
- Deque is a linear data structure where the insertion and deletion operations are performed from both ends.
- We can say that deque is a generalized version of the queue.

DEQUES (Double ended queue)

- Though the insertion and deletion in a deque can be performed on both ends, it does not follow the FIFO rule. The representation of a deque is given as follows –



Representation of deque

DEQUES (Double ended queue)

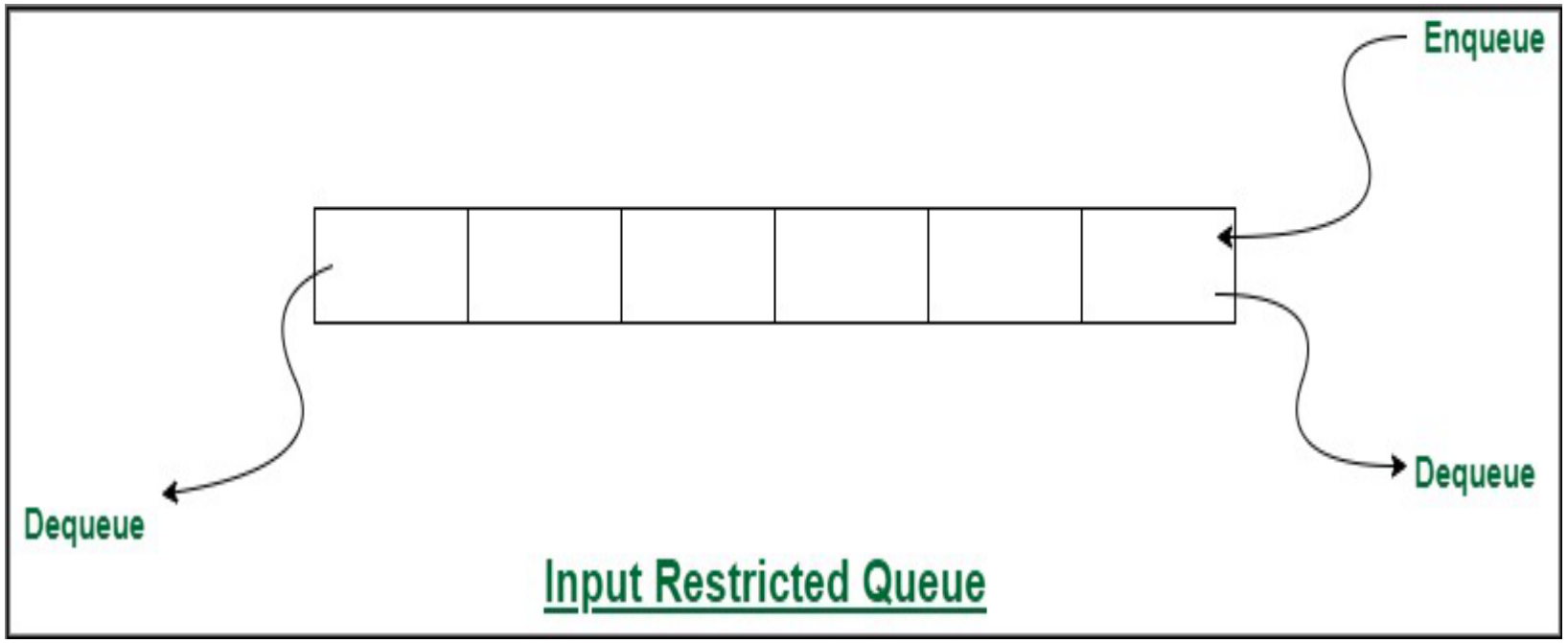
Types of deque

There are two types of deque -

1. **Input restricted deque:** An input restricted deque is a deque, which allows insertion at only 1 end, rear end, but allows deletion at both ends, rear and front end of the lists.
2. **Output restricted deque:** An output-restricted deque is a deque, which allows deletion at only one end, front end, but allows insertion at both ends, rear and front ends, of the lists.

DEQUES (Double ended queue)

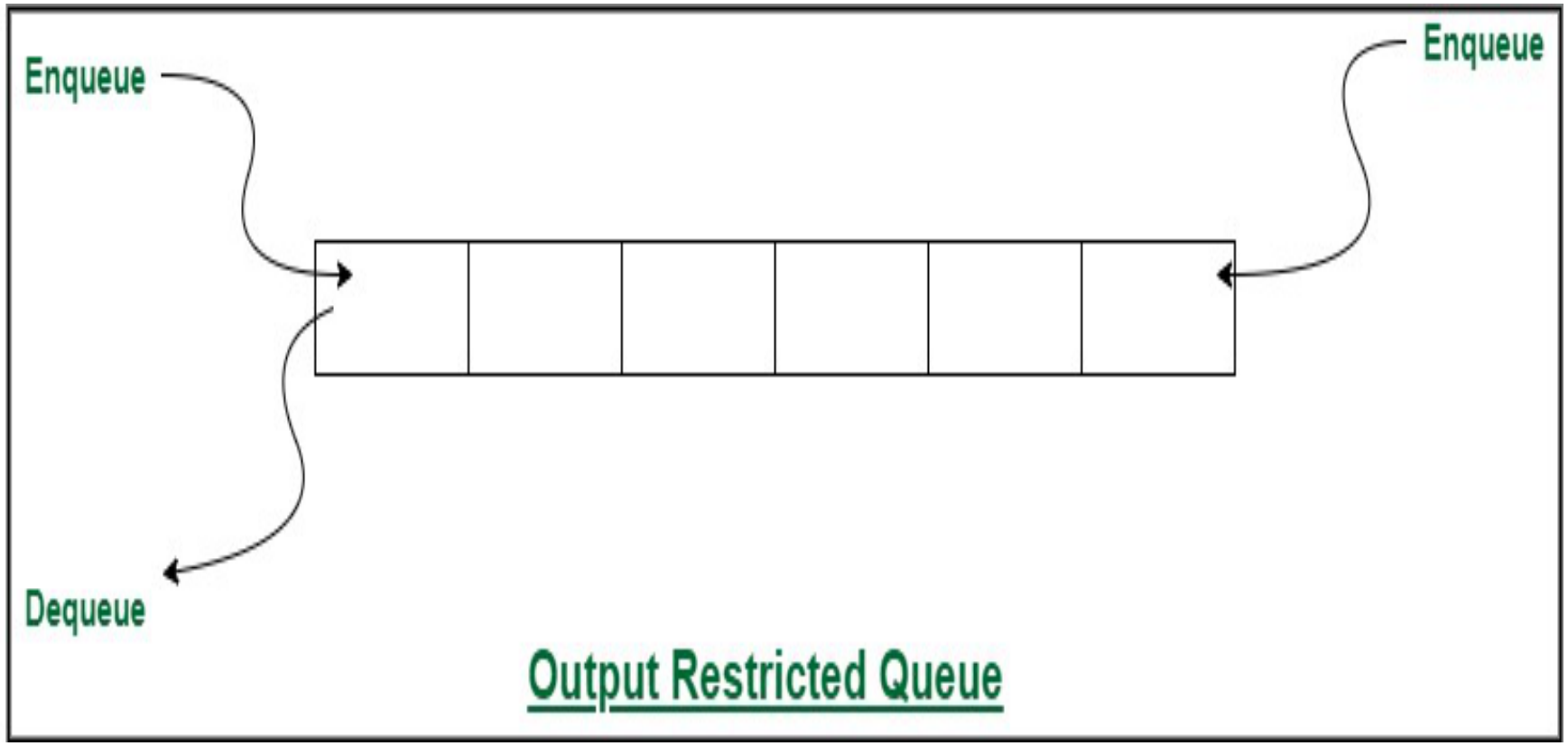
- **In input restricted queue**, insertion operation can be performed at only one end, while deletion can be performed from both ends.



DEQUES (Double ended queue)

Output restricted Queue

- **In output restricted queue**, deletion operation can be performed at only one end, while insertion can be performed from both ends.



DEQUES (Double ended queue)

Operations performed on deque

There are the following operations that can be applied on a deque:

- Insertion at front
- Insertion at rear
- Deletion at front
- Deletion at rear

DEQUES (Double ended queue)

The possible operation (Deque ADT) performed on deque is

1. Add an element at the rear end (insert_rear)
2. Add an element at the front end (insert_front)
3. Delete an element from the front end (delete_front)
4. Delete an element from the rear end (delete_rear)

Note: Only 1st, 3rd and 4th operations are performed by input-restricted deque and 1st, 2nd and 3rd operations are performed by output-restricted deque.

Algorithms For Inserting An Element

- Let Q be the array of MAX elements.
- front (or left) and rear (or right) are two array index (pointers), where the addition and deletion of elements occurred.
- Let DATA be the element to be inserted.
- Before inserting any element to the queue left and right pointer will point to the -1.

Algorithms to Insert an element at the right side (rear end) of the de-queue

1. Input the DATA to be inserted
2. If ((front == 0 && rear == MAX–1) | | (front == rear + 1))
   i. Display “Queue Overflow”
   ii. Exit
3. Elseif(front == –1 && rear == -1)
   i. front = rear = 0
   ii. Q[rear] = DATA
4. Else
   i. if (rear != MAX –1) //Deque already contain more than one element
      i. rear = rear+1
      ii. Q[rear] = DATA
   ii. Else //rear has reach at the end of array
      [Shift all the elements from position front until rear back by 1 position]
      for(i = front; i<=rear;i++)
      Q[i-1] = Q[i]
      Q [rear] = DATA
5. Exit

Insert An Element At The Left Side (Front) Of The De-Queue

1. Input the DATA to be inserted
2. If ((front == 0 && rear == MAX–1) | | (front == rear+1))
   i. Display “Queue Overflow”
   ii. Exit
3. ElseIf (front == – 1 && rear == -1)
   i. front = rear = 0
   ii. Q [front] = DATA
4. Else
   i. if (front != 0) // Deque already contain more than one element
      a. front = front– 1
      b. Q [front] = DATA
   ii. Else
      a. Shift all the elements from position front until rear ahead by 1 position

      for(i=rear; i>=front;i--)

            Q[i+1] = Q [i]

      a. Q[front] = DATA
5. Exit

ALGORITHMS FOR DELETING AN ELEMENT

- Let Q be the array of MAX elements. front (or left) and rear (or right) are two array index (pointers), where the addition and deletion of elements occurred.
- DATA will contain the element just deleted.

Algorithm Delete An Element From The Right Side (Rear side) Of The De-Queue

1. If (front == – 1 && rear == -1)
    i. Display “Queue Underflow”
    ii. Exit.
2. If (front == rear) //Deque has only one element (last element)
    i. DATA = Q [rear]
        a. front = – 1
        b. rear = – 1
3. Else //deque has more than one element
    i. DATA = Q [rear]
        a. rear = rear -1
4. Exit

Algorithm: Delete An Element From The Left Side (Front side) Of The De-Queue

1. If (front == – 1 && rear == -1)
    i. Display “Queue Underflow”
    ii. Exit
2. ElseIf(front == rear)
    i. DATA = Q [front]
        a. front = – 1
        b. rear = – 1
3. Else // Dequeue has more than one element
    i. DATA = Q [front]
    ii. Front = front +1
4. Exit

Priority Queue

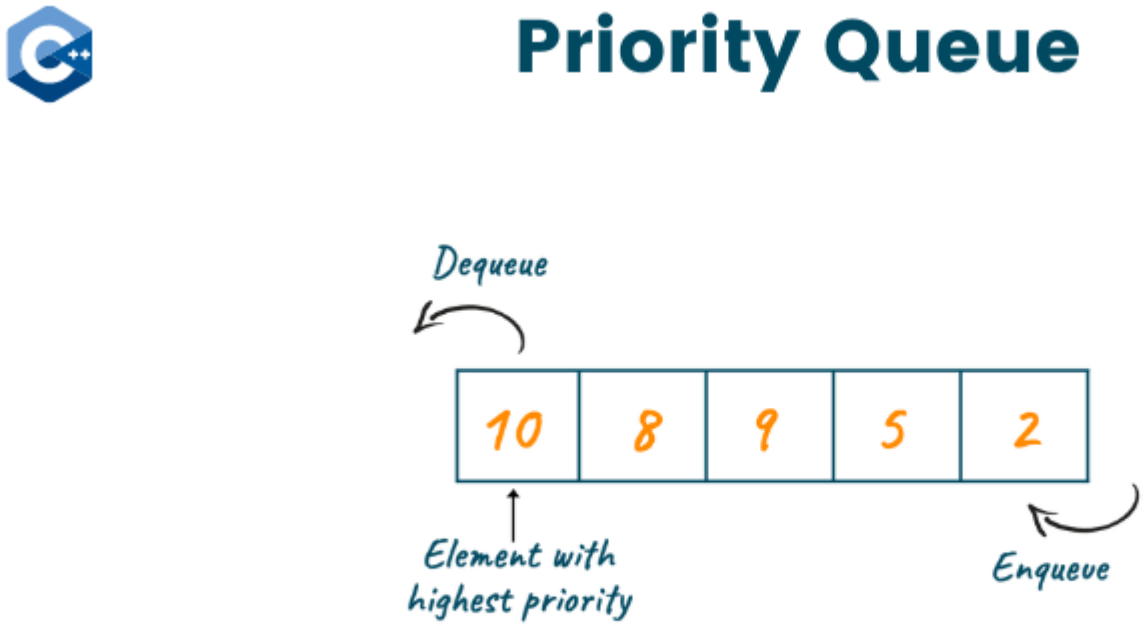
Priority Queue is a queue where each element is assigned a priority. In priority queue, the elements are deleted and processed by following rules:

I. An element of higher priority is processed before any element of lower priority.

II. Two elements with the same priority are processed according to the order in which they were inserted to the queue.

Priority Queue

1. Assignment Priority Queue!!



APPLICATION OF QUEUES

1. Round robin techniques for processor scheduling is implemented using queue.
2. Printer server routines (in drivers) are designed using queues.
3. All types of customer service software (like Railway/Air ticket reservation) are designed using queue to give proper service to the customers.
4. Disk Driver: maintains a queue od disk input/output requests
5. Scheduler (e.g, in operating system): maintains a queue of processes awaiting a slice of machine time.
6. Call center phone systems will use a queue to hold people in line until a service representative is free.

APPLICATION OF QUEUES

1. Buffers on MP3 players and portable CD players, iPod playlist. Playlist for jukebox - add songs to the end, play from the front of the list.
2. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
3. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Thank you!!!!!