Chapter 3
QUEUE

Data Structure & Algorithm

Compiled by :

Ankit Bhattarai
Email: ankit.bca@kathford.edu.np

Syllabus

Unit	Contents	Hours	Remarks
1.	Introduction to Data Structure	2	
2.	The Stack	3	
3.	Queue	3	
4.	List	2	
5.	Linked Lists	5	
6.	Recursion	4	
7.	Trees	5	
8.	Sorting	5	
9.	Searching	5	
10.	Graphs	5	
11.	Algorithms	5	

Credit : 3

Outlines

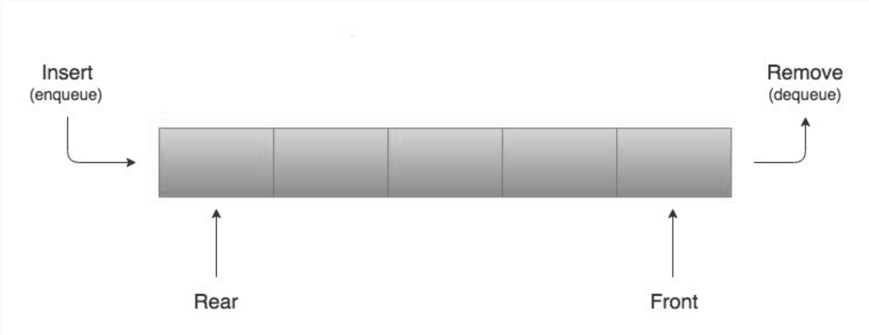
- Introduction
- Queue as an ADT
- Primitive operations in Queue
- Linear and Circular Queue and their applications
- Enqueue and Dequeue
- Priority Queue

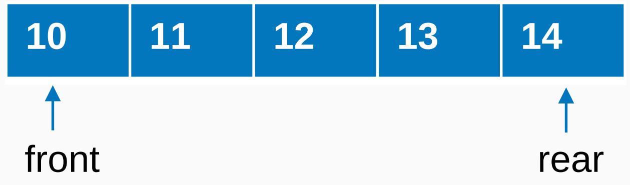
Introduction to QUEUE

- A queue is a linear data structure . It is an ordered collection of items in which items are inserted at one end called **rear** and existing items are deleted at other end called **front**.
- It is also called the First In First Out type of data structure (FIFO), since the first item removed is the first item inserted.
- Scenario in real world:

✓ Printing of files in a network of computers.

✓ Queues at theatres (movie tickets).

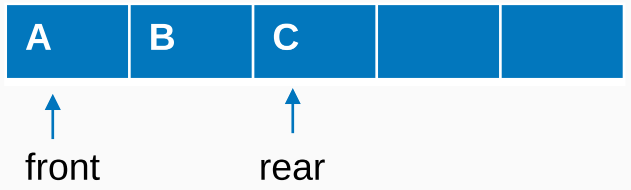




Array size = 5

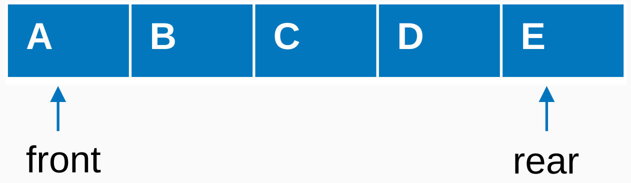
Example of QUEUE

- A queue containing three elements A, B and C with two ends.

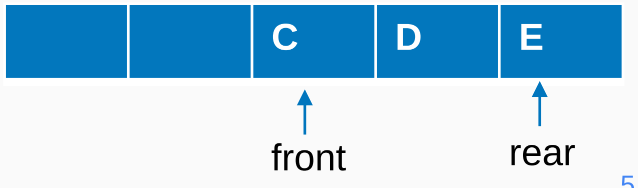


Array size = 5

- If D and E are to be inserted then the queue will look like:

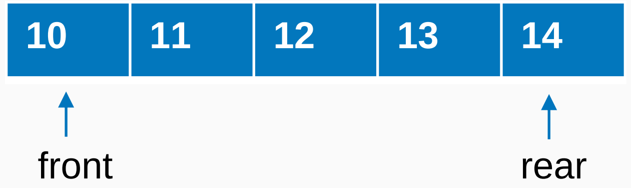


- If we want to remove B, it is not possible until A is in the queue, Because B is not the front. So we have to remove A & then B.



QUEUE Applications

- Queue is most often used in a scenario where there is a shared resource that is supposed to serve some request, but the resource can handle only one request at a time.

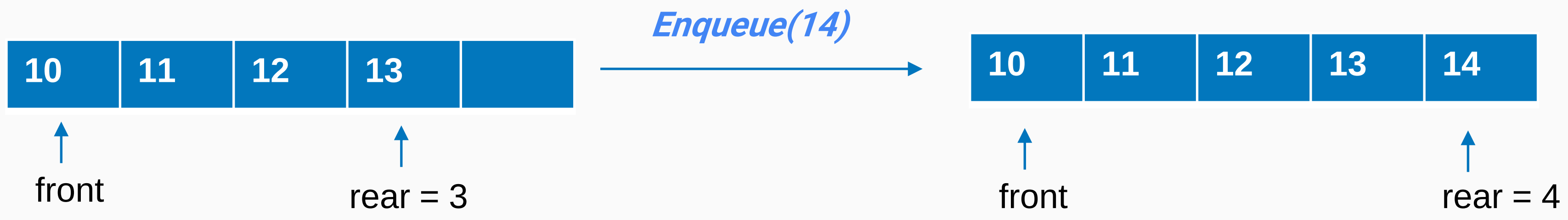


- **Applications of Queue:**

1. Print queue: Jobs sent to the printer.
2. Operating System maintains queue in allocating the process to each unit by storing them in buffer.
3. Queue acts as a auxiliary data structure for various algorithms and it is component of other data structures.

Primitive Operations in QUEUE

- **Enqueue** (Insert operation)
    - Inserts an item at the rear of the queue. Other names - Add, Insert
    - The rear of the queue will be now occupied with an item currently added in the queue.
    - Rear count will be incremented by one after addition of new data item.
    - **rear = rear + 1**



Primitive Operations in QUEUE

- **Dequeue** (Delete operation)
  - Deletes an item from the front of the queue. Other names – Delete, Remove
  - Now the front will be incremented by one each time when an item is removed from the queue.
  - **front = front + 1**

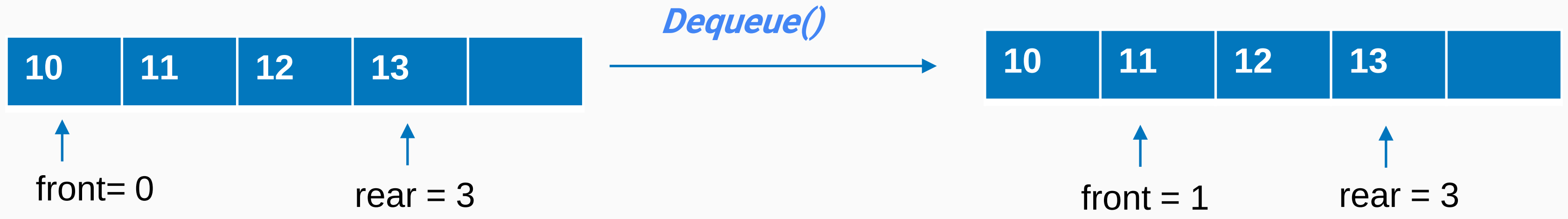
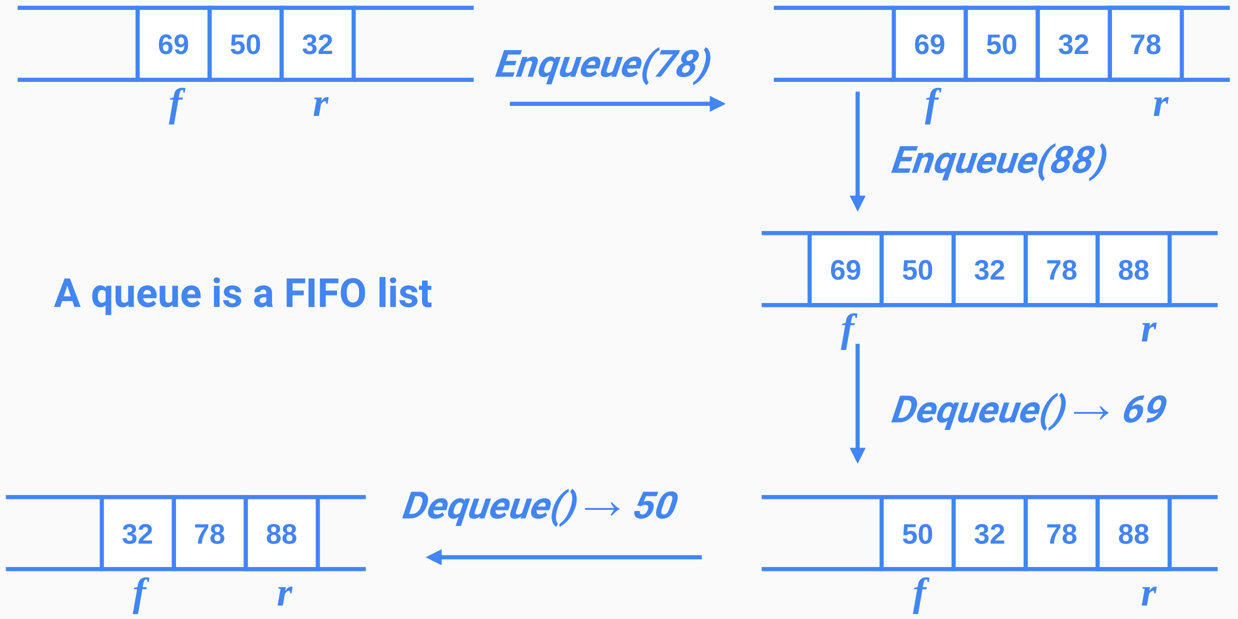


Illustration of Queue Operation



A queue is a FIFO list

QUEUE as an ADT

Values: A queue of elements of type T is a finite sequence of elements of T together with the operations

Operations:

1. **MakeEmpty(Q)** : Create an empty queue Q
2. **Empty(Q)** : Determine if the queue Q is empty or not
3. **Enqueue(Q, x)** : Insert element x at the end of the queue Q
4. **Dequeue(Q)** : If the queue Q is not empty, remove the element at the front of the queue
5. **Front(Q)** : Retrieve the element at the front of the queue Q, without deleting it
6. **Full(Q)** : Checks if the queue is full or not.

Implementation of Queue

1. **Static Implementation Using Array**

- Linear array: A linear array, is a list of finite numbers of elements stored in the memory. In a linear array, we can store only homogeneous data elements.

- Circular array : An array is called circular if we consider the first element as next of the last element.

2. **Dynamic Implementation using Linked List** (Chapter 5)

Types of Queue

i. Linear Queue

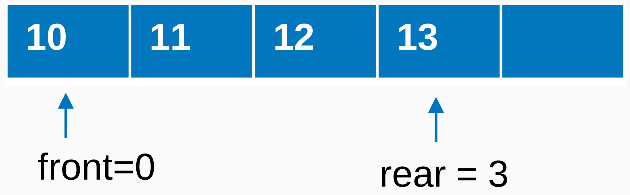
ii. Circular Queue

iii. Priority Queue

iv. DEQUE (Double Ended Queue)

i. Linear Queue:

A linear queue is a linear data structure that serves the request first, which has been arrived first. It consists of data elements which are connected in a linear fashion.



Algorithm for enqueue operation in Linear queue

```
Step 1: Start
Step 2: Check if the queue is full or not
        if the queue is full write " Queue overflow or queue is full"
        and go to step 5
Step 3: If the queue is not full increment
        rear ← rear + 1
        to the next empty space.
Step 4: Add the data item in the queue where the rear is pointing.
        queue[rear] ← item
Step 5: Stop
```

Algorithm for dequeue operation in Linear queue

```
Step 1: Start
Step 2: Check if the queue is empty or not
        if the queue is full write " Queue underflow or queue is empty"
        and go to step 5
Step 3: If the queue is not empty access the data item where front is pointing
                queue[front] = item
Step 4: Increment the front pointer to the next data items.
                front = front + 1
Step 5: Stop
```

Or, Algorithm for enqueue operation in Linear queue

Queue is declared as int queue [MaxSize], front = -1 and rear = -1

```
Step 0: Start

Step 1: Initialization set front = -1, set rear = -1

Step 2: Repeat steps 3 to 5 Until rear < MaxSize -1

Step 3: Read item

Step 4: if front == -1 then
                    front = 0
                    rear = 0
          else
                    rear = rear + 1
          endif
Step 5: Set queue[rear] = item
Step 6: Print Queue is overflow
Step 7: Stop
```

Or, Algorithm for dequeue operation in Linear queue

Queue is declared as int queue [MaxSize], front = -1 and rear = -1

```
Step 0: Start
Step 1: Repeat steps 2 to 4 until front > = 0
Step 2: set item = queue [ front]
Step 3: if front == real then
                    set front = -1
                    set rear = -1
          else
                    front = front + 1
          endif
Step 4: Print " Deleted item is "
Step 5: Print " Queue underflow or queue is empty"
Step 6: Stop
```

Question to practice

Consider the following linear queue capable of maintaining upto 10 elements.

front = 0, rear = 2

elements = A, B, C

Describe the queue as the following operations takes place.

i. F is added to the queue.
ii. Two letters are added.
iii. K, L, M are added to the queue.
iv. One letter is deleted.
v. R is added to the queue.

Show front & rear value on each of the operation.

Question to practice

Given,

A	B	C							
f=0		r=2							

i.

A	B	C	F						
f=0			r=3						

ii.

		C	F						
		f=2	r=3						

iii.

		C	F	K	L	M			
		f=2				r=6			

iv.

			F	K	L	M			
			f=3			r=6			

v.

			F	K	L	M	R		
			f=3				r=7		

Program to implement Enqueue operation in Queue using array

Algorithm for Enqueue operation

Queue is declared as int queue [MaxSize], front = -1

and rear = -1

```
Step 0: Start
Step 1: Initialization set front = -1, set rear = -1
Step 2: Repeat steps 3 to 5 Until rear < MaxSize -1
Step 3: Read item
Step 4: if front == -1 then
                front = 0
                rear = 0
            else
                rear = rear + 1
            endif
Step 5: Set queue[rear] = item
Step 6: Print Queue is overflow
Step 7: Stop
```

Module/Code for Enqueue operation

```
void enqueue()
{
if ( rear < n -1)
   {
            printf(" Enter the data item to be inserted \n");
            scanf("%d", &item);
            if (front== -1)
            {
                        front = 0;
                        rear = 0;
                        }
                        else
                        {
                         rear= rear + 1;
                        }
                        queue[rear] = item;

            }
else
    printf(" Queue is full or queue overflow \n");
}
```

Program to implement Dequeue in Queue using array

Algorithm for Dequeue operation

Queue is declared as int queue [MaxSize], front = -1

and rear = -1

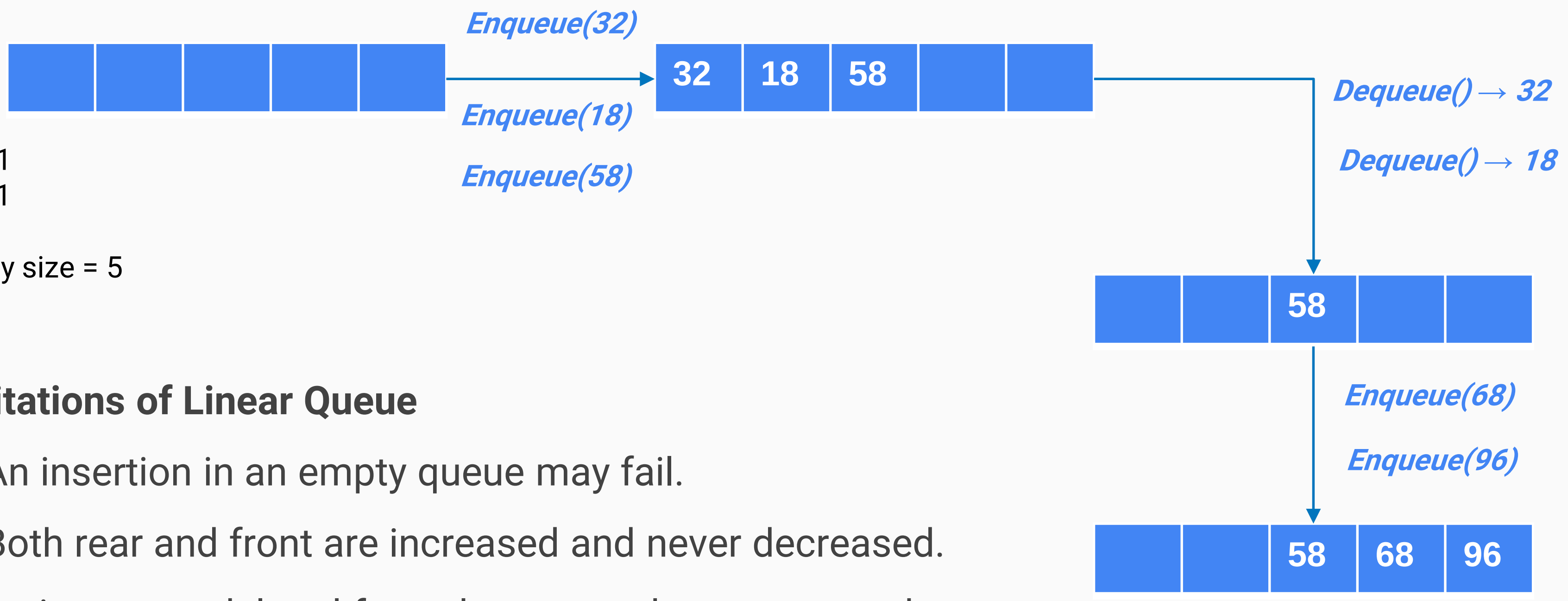
```
Step 0: Start
Step 1: Repeat steps 2 to 4 until **front > = 0**
Step 2: set **item = queue [ front]**
Step 3: if front == real then
                set front = -1
                set rear = -1
            else
                front = front + 1
            endif
Step 4: Print " Deleted item is "
Step 5: Print " Queue underflow or queue is empty"
Step 6: Stop
```

Module/Code for Dequeue operation

```
void dequeue()
{
   if (front != -1)
   {
                  item = queue[front];
                  if (front==rear)
                        {
                            front = -1;
                             rear = -1;
                          }
                  else
                      front = front + 1;
                      printf(" Deleted data item is %d \n",item);
                  }
    else
        printf("Queue is empty or queue underflow \n");
}
```

Illustration of Limitation of Linear Queue

Illustration:



f = -1
r = -1

Array size = 5

Nothing can be added in the queue now.

Limitations of Linear Queue

- An insertion in an empty queue may fail.
- Both rear and front are increased and never decreased.
- As items are deleted from the queue, the storage at the beginning of the array is discarded & never used again.

Linear Queue: Limitations & their possible solutions

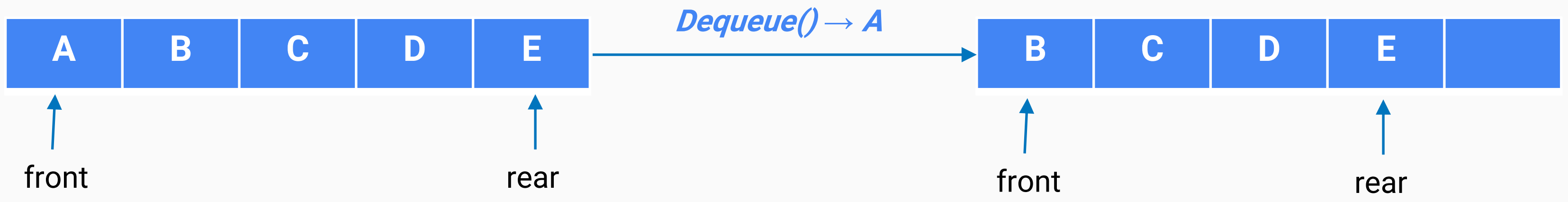
Possible Solutions

1. Shifting
2. Circular Queue

Explanation

1. Shifting:

- Whenever we remove an item from the queue, we may also shift other items in the queue to the beginning of the array as shown below:



Linear Queue: Limitations & their possible solutions

✓ **Algorithm for shifting**

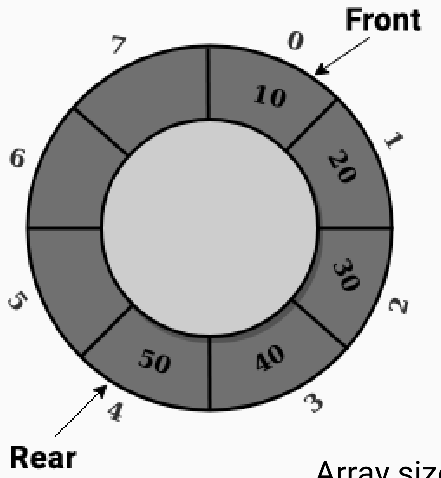
```
begin procedure
    use loop
        shift the element
        rear - -
stop
```

- Under this representation, we do not require the front field, since the element at position zero of the array is always the front of the queue.
- But it is inefficient because each removal operation involves moving every remaining element of the queue.
- If a queue contains 1000 elements, then shifting operation would consume a lot of time logically removing an item involves manipulation of only one element but this solution involves numerous shifting operation in addition.

Linear Queue: Limitations & their possible solutions : Circular Queue

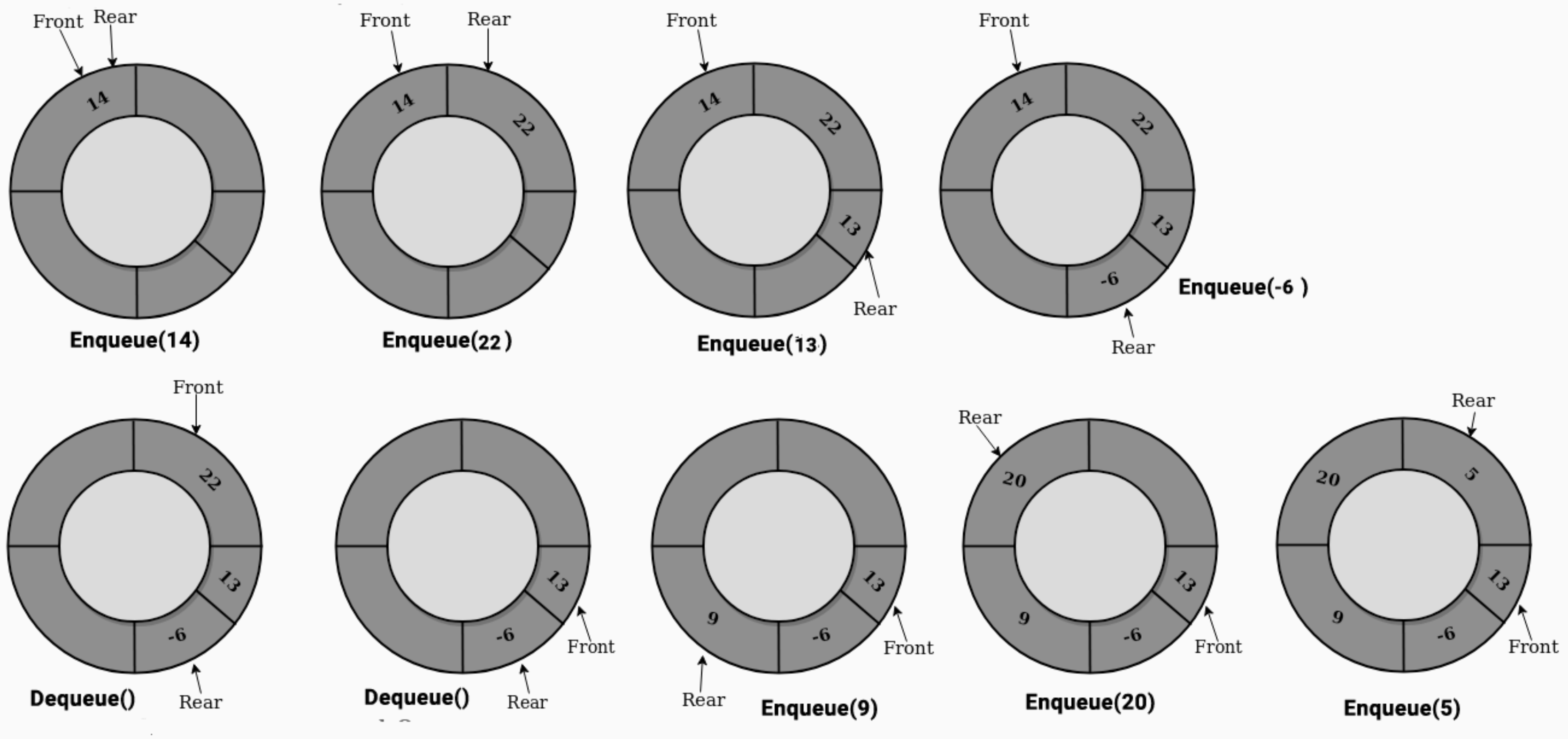
2. Circular Queue

- Another solution to overcome the limitation of linear queue is to view the array that holds the queue as a circular.
- The first element of the array follows the last element.
- This way as the data items are added & removed from the queue, the rear will always chase the front around the array.
- So, the items can be inserted unless array is fully occupied.
- Circular queue is a linear data structure.
- It follows as FIFO principle.



Array size = 8

Illustration of Circular Queue



Algorithm for enqueue operation in Circular queue

```
Step 1: Start
Step 2: if ( front == (rear + 1) % maxsize)
                    write " Queue overflow or queue is full"
       else
                    Read the value to be inserted
                    if (front == -1)
                              set front = rear = 0
                    else
                              rear = ((rear + 1) % maxsize)
                              assign  queue[rear] = value
        endif
Step 5: Stop
```

Algorithm for dequeue operation in Circular queue

```
Step 1: Start
Step 2: if ( front == -1)
                write " Queue underflow or queue is empty"
        else
                item = queue [front]
                if (front == rear)
                        set front = -1
                        set rear = -1
                else
                        front = ((front + 1) % maxsize)
         endif
Step 5: Stop
```

Program to implement Enqueue operation in Circular Queue using array

Algorithm for Enqueue operation

```
Step 1: Start
Step 2: if ( front == (rear + 1) % maxsize)
                write " Queue overflow or queue is full"
        else
                Read the value to be inserted
                if (front == -1)
                    set front = rear = 0
                else
                    rear = ((rear + 1) % maxsize)
                    assign  queue[rear] = value
        endif
Step 5: Stop
```

Module/Code for Enqueue operation

```
void enqueue()
{
            if ( front == (rear + 1) % maxsize)
            {
              printf("Circular Queue overflow or Circular queue is full \n");
            }
           else
            {
                        printf(" Enter the data item to be inserted \n");
                        scanf("%d", &item);
                        if (front == -1)
                        {
                        front = rear = 0;
                        }
                        else
                        {
                        rear = ((rear + 1) % maxsize);
                        }
                        cqueue[rear] = item;
           }
}
```

Program to implement Dequeue operation in Circular Queue using array

Algorithm for Dequeue operation

```
Step 1: Start
Step 2: if ( front == -1)
                  write " Queue underflow or queue is empty"
        else
              item = queue [front]
              if (front == rear)
                  set front = -1
                  set rear = -1
              else
                  front = ((front + 1) % maxsize)
          endif
Step 5: Stop
```

Module/Code for Dequeue operation

```
void dequeue()
{
  if ( front == -1)
                 {
                   printf("Circular Queue underflow or Circular
queue is empty \n");
                 }
                 else
                  {
                   item = cqueue[front];
                   printf("Deleted element is %d \n", item);
                   if (front == rear)
                                   front= rear = -1;
                    else
                                   front = ((front + 1)% maxsize);
                 }
}
```

Priority Queue

- In queues, the items are ordered based on the sequence in which they are inserted.
- But in some cases, we need that items of queue to serviced out of order of insertion i.e. the queue items are accessed or manipulated according to associated priority number.
- For such functionality, we need special kind of queue called priority queue.
- A **priority queue** is a data structure in which items are manipulated & deleted according to the following rules:
    - ➢ The elements having highest priority is processed before any element of lower priority.
    - ➢ If the elements have equal priority then they get manipulated according to the order in which they are inserted in priority order.

Priority Queue

- **Types of priority queue**
  - Ascending (Minimum) Priority Queue:

    It is a collection of items in which items can be inserted arbitrarily but only the smallest item is removed first.
  - Descending (Maximum) Priority Queue:

    It is a collection of items in which items can be inserted arbitrarily but only the smallest item is removed first

Priority Queue as an ADT

Values: A minimum priority queue contains finite elements of type T.

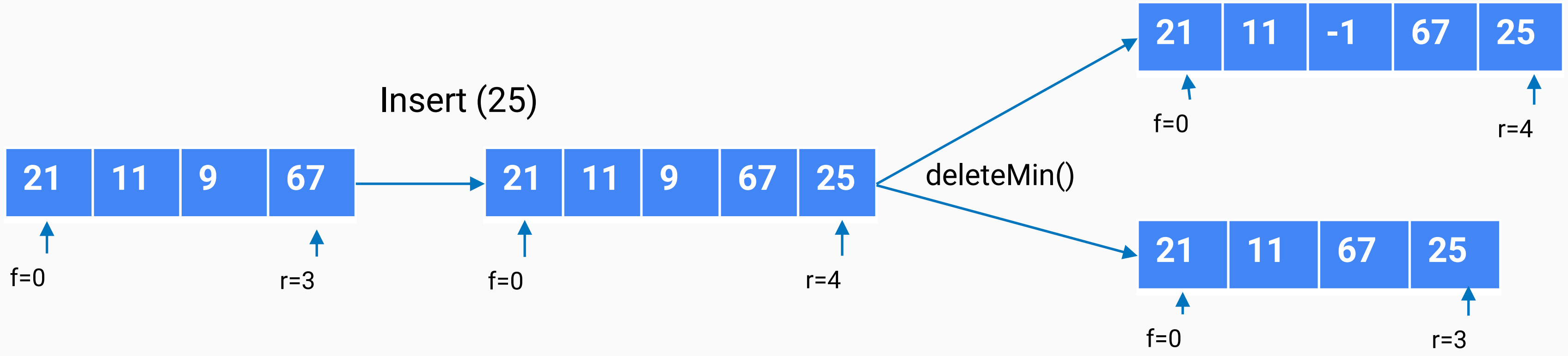
Operations:

1. makeEmpty(P): creates an empty priority queue P.
2. Empty(P):checks if the priority queue is empty or not.
3. Enqueue(P,x): Add the element x on the priority queue P.
4. deleteMin(P): If the priority queue P is not empty, it removes the minimum element of the queue & returns it.
5. FindMin(P): Retrieve the minimum element of the queue & return it without removing the element.

Array Implementation of priority queue [For minimum]

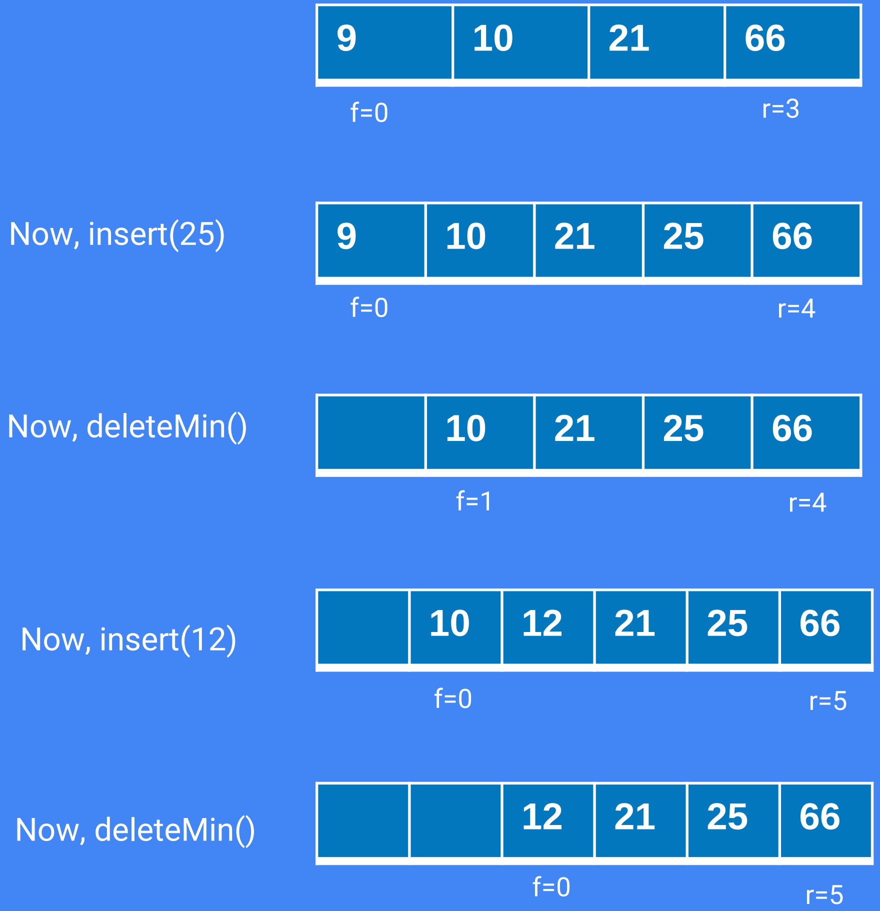
a. **Unordered Array Implementation**

- Insertion is done at the rear end of the queue.
- For deletion, the position of the minimum element is first found out, then either
  - ✓ A special 'empty' indicator (like -1) can be placed into the deleted position. (This method is called lazy deletion).
  - ✓ Or, We shift all the elements past the deleted element by one position & then decrease the rear by 1.



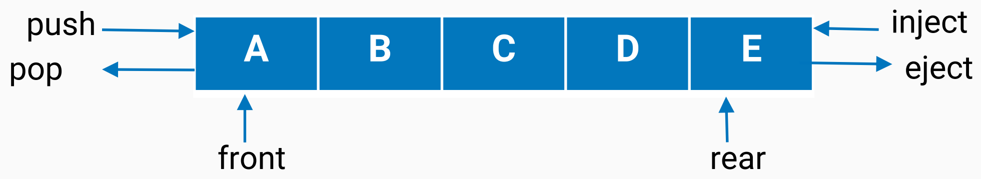
b. **Ordered Array Implementation**

- Instead of maintaining the queue as an unordered array we maintain it as a circular ordered array making the front as position of smallest element.
- For insertion, we locate the proper position of the new element & then we shift the succeeding element by 1 position.
- For deletion, we delete the minimum element & increase the front position.



DEQUE (Double Ended Queue)

- **DEQUE** is a data structure in which insertion & deletion operations are performed at both ends.
- We can insert items at rear end or at the front end. Also, we can delete from rear or front end.



- **Operations:**

✓ Push(dq,x): Insert item x at the front of deque(dq).

✓ Pop(dq): Remove the front item from dq.

✓ Inject(dq,x): Insert the item x at rear end of dq.

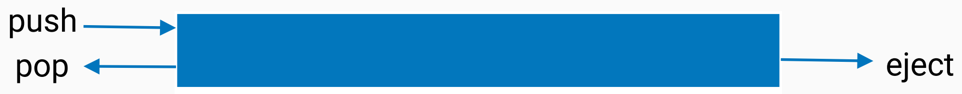
✓ Eject(dq): Remove the rear item from deque dq and return it.

DEQUE (Double Ended Queue)

- **DEQUE** can be divided into two types. On the basis of restriction put to perform either the insertion or deletion at one end.

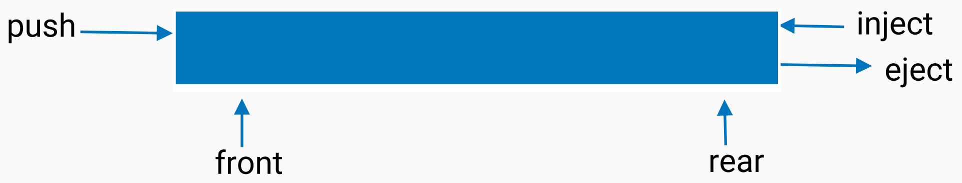
1. **Input restricted deque** (input one, two output)

- For this type of deque, only the operations push, pop & eject are valid.



2. **Output restricted deque** (input two, output one)

- For this type of deque, only the operations push, inject & eject are valid



Differences between Stacks & Queues

Stacks	Queues
Stack is an ordered list where in all insertions and deletions are performed at one end called top of the stack(TOS).	Insertions are performed at rear end and deletions are performed at other end called front.
It follows LIFO (Last In First Out) structure	It follows FIFO(First In First Out) structure
To insert an element into the stack top of the stack is incremented by 1	To insert an element into the queue rear is incremented by 1.
To delete an element into the stack top of the stack is decremented by 1	To delete an element into the queue front is incremented by 1

Question to focus

1. Define linear and circular queue. What is queue overflow and underflow?
2. Write down the algorithm to perform enqueue & dequeue operation in queue (or linear queue).
3. Differentiate between Stack and Queue.
4. With the help of an example, compare circular queue with linear queue.
5. What are the limitations of linear queue & the possible solutions to overcome the limitations?
6. Write short notes on:

a. Limitation of linear queue
b. Queue as an ADT
c. Priority queue
d. Deque (Double ended queue)

Question to focus

6. Write down the application of queue.

7. Distinguish between linear queue & circular queue on the basis of its operation technique.

8. Write down the module for enqueue operation in queue.

9. Write down the module for dequeue operation in queue.

9. How can you perform Enqueue and Dequeue operation in circular queue? State its algorithm.

10. What is priority queue? Explain array implementation of priority queue.

11. Suppose you are asked to arrange people waiting for tickets in a cinema hall which data structure would you use to make the arrangement. Discuss suitable algorithm for your arrangement.

Question to focus

12. Consider the following queue characters where Queue is a circular array which is allocated in size 6:

Front = 2, Rear = 4, Queue elements: A, C, D

i. F is added in the queue
ii. Two letters are deleted
iii. K, L, M are added.
iv. Two letters are added.
v. S is added to the queue
vi. One letter is deleted
vii. R is added to the queue
viii. One letter is deleted.

THANK YOU

Any Queries ?

ankit.bca@kathford.edu.np