Data Structures
Module 2.3
Stack and Queues

Ms. Sarika Dharangaonkar
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Introduction to Queue
- Queue ADT
- Representation of Queue in memory
- Operations on Queue
- Algorithms for Queue operations
- Implementation of Queue

Learning Objectives

At the end of the lecture, students will be able to

- Demonstrate different operations performed on a queue.
- Develop program to implement queue ADT using array and linked list.

Queue

- A queue is an Abstract Data Type (ADT), commonly used in most programming languages.

- It is named queue as it behaves like a real-world queue, for example-

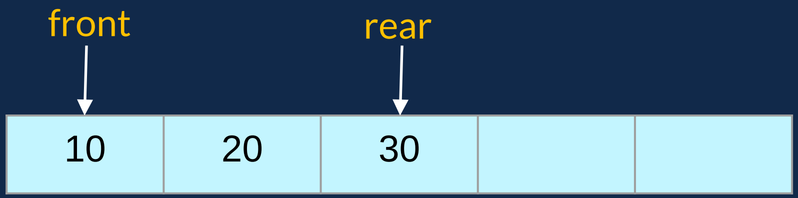
Queue at the Airport

LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT

Queue at the Signal

Queue

- A queue is a collection of elements, which can be stored and retrieved one at a time.
- In queue, element is inserted from one end called the rear end and deleted from another end called the front end of the queue.
- The first element inserted in the queue is the first element removed from the queue . Hence, queue is a First-in First-out data structure(FIFO).
- The queue is known as restrictive data structure because the operations are restricted to the front and rear end of the structure.



Queue ADT

Operation	Description
enqueue(x)	This is used to insert x at the rear end of the queue
dequeue()	This is used to delete one element from front of the queue
isFull()	This is used to check whether queue is full or not
isEmpty()	This is used to check whether queue is empty or not
size()	This function is used to get number of elements present into the queue

Implementation of Queue

Queue data structures are usually implemented using arrays or linked lists.

1. Static Implementation using Arrays :

For applications in which the maximum queue size is known ahead of time, an array is suitable.

2. Dynamic implementation using linked List:

If the maximum queue size is not known beforehand, we could use a linked list

Array Representation of Queue

- Queue can be represented as an array in the computer memory.
- To store the address of the front element of the queue, every queue has a variable called front associated with it.
- To store the address of the last element of the queue, every queue has a variable called rear associated with it.
- Another variable called MAX is used which represent the maximum number of elements that the queue can hold.

- Example:

```
#define MAX 20
int queue[MAX], front=0, rear=-1;
```

Initialize Queue using Array

front = 0
rear = -1

0	1	2	3	4

```
int queue[5];

int front=0, rear=-1;
```

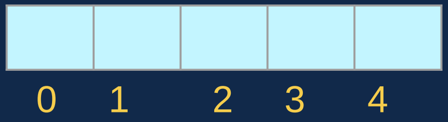
First element 10 inserted in queue

front = rear = 0

10				
0	1	2	3	4

Enqueue Operation

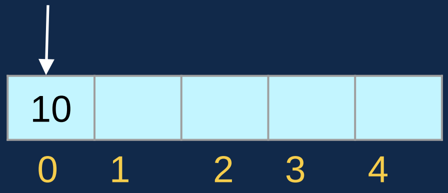
front = 0 rear = -1



0	1	2	3	4

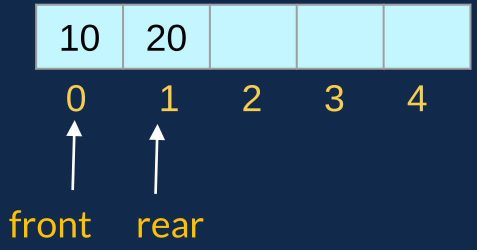
Insert 10 in empty queue

front = rear = 0



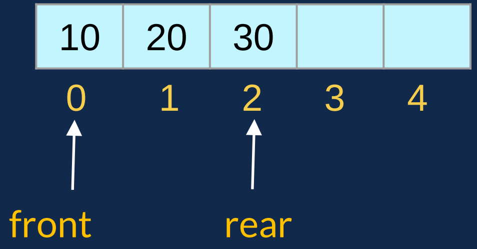
10				
0	1	2	3	4

Insert 20 in queue



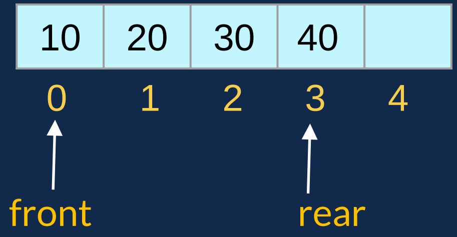
10	20			
0	1	2	3	4

Insert 30 in queue



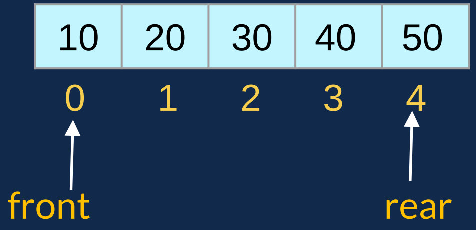
10	20	30		
0	1	2	3	4

Insert 40 in queue



10	20	30	40	
0	1	2	3	4

Insert 50 in queue



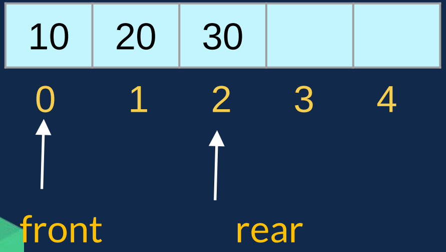
10	20	30	40	50
0	1	2	3	4

Enqueue Operations on a Queue

Algorithm: enqueue operation

```
Step 1: IF rear = MAX-1 then
            PRINT OVERFLOW & STOP
Step 2: SET rear = rear + 1
Step 3: SET queue[rear] = data
Step 4: END
```

dequeue Operation

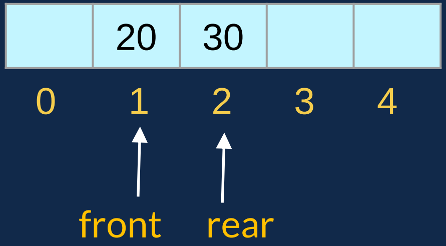


0	1	2	3	4
10	20	30		

front

rear

Delete element from queue

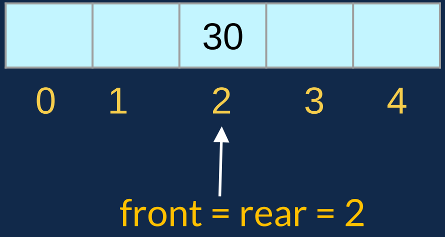


0	1	2	3	4
	20	30		

front

rear

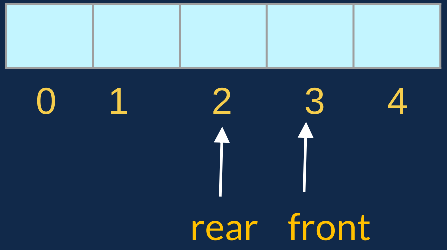
Delete element from queue



0	1	2	3	4
		30		

front = rear = 2

Delete element from queue



0	1	2	3	4

rear

front

Dequeue Operation on a Queue

Algorithm: dequeue operation

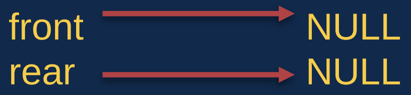
```
Step 1: IF front > rear
            PRINT UNDERFLOW & STOP
Step 2: SET data = queue[front]
Step 3: SET front= front + 1
Step 4: RETURN data
Step 5: END
```

Linked Representation of Queue

- Queue can be represented using linked list in the computer memory.
- In linked queue, every element has two parts, one that stores the data and another that stores the address of the next element.
- The first node of linked list is pointed by front.
- The last node of the linked list will be pointed by rear.
- All insertion will be done at the rear end and all deletion will be done at the front end.
- If front=rear=NULL , then it indicates that the queue is empty.

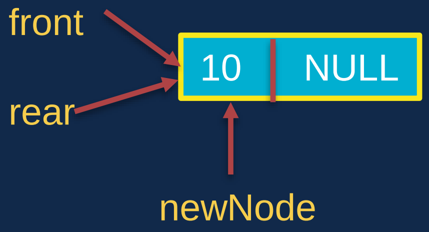
Initialize Queue using Linked Representation

```
Struct node
{
                int data;
                struct node *next;
};
typedef struct node Node;
Node *front,*rear;
front=rear=NULL;
```

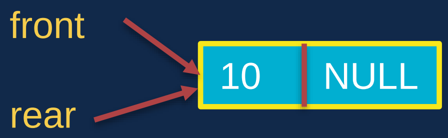


First element 10 inserted in queue

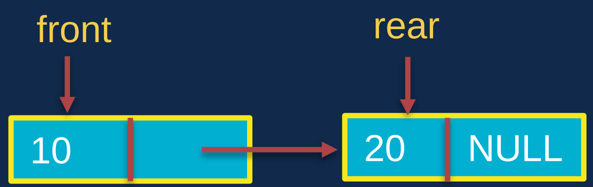
```
                enqueue(10);
                     ↓
Node *newNode;
newNode=(Node*)malloc(sizeof(Node);
newNode->data=10;
newNode->next=NULL;
If(front==NULL && rear==NULL)
                front=rear=newNode;
```



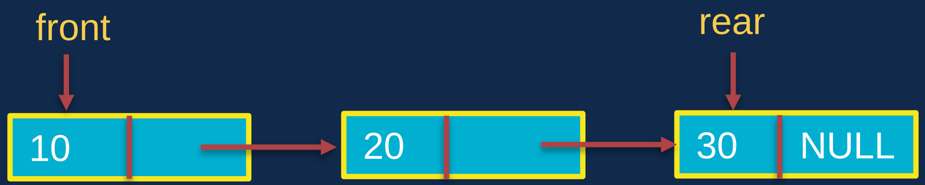
Enqueue operation in Linked Queue



Insert 20 in queue



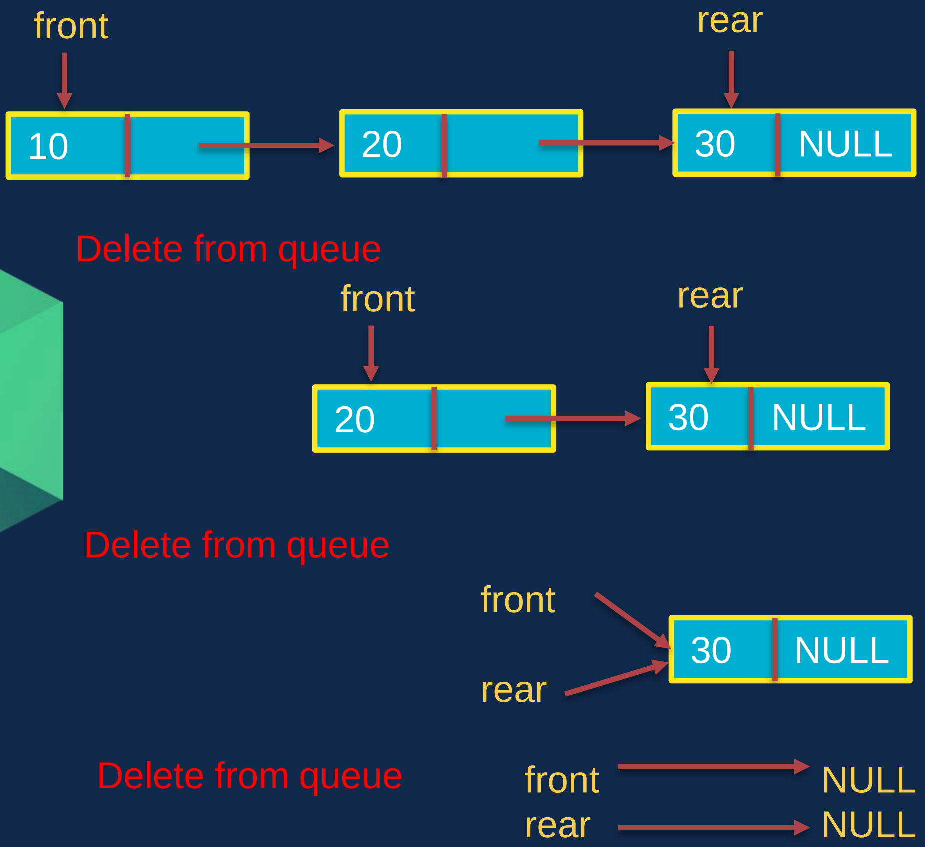
Insert 30 in queue



Algorithm: enqueue operation

```
Step 1: If queue isEmpty
            SET front = rear = newNode
        else
            SET rear -> next = newNode
            SET rear = newNode
Step 2: END
```

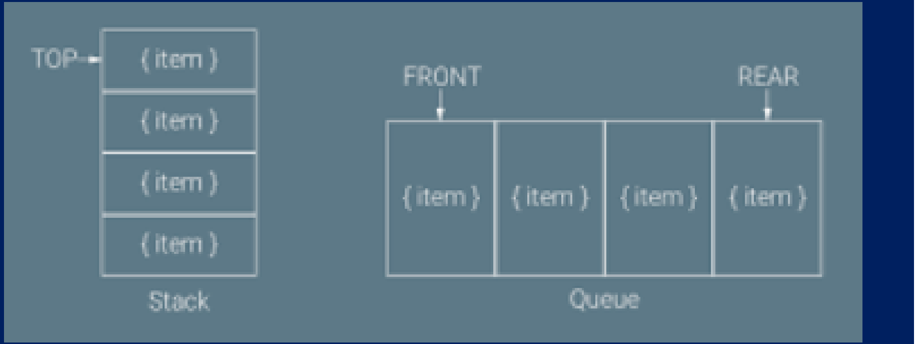
Dequeue operation in Linked Queue



Algorithm: dequeue operation

```
Step 1: IF  queue isEmpty
            PRINT UNDERFLOW & STOP
Step 2:  SET temp = front
Step 3:  SET value = front->value
Step 4: if front = rear
            SET front = rear = NULL
        else
            SET front = front->next
Step 5: free(temp)
Step 4: RETURN value
```

Stack Queue



Queue

- First In First Out(FIFO List)
- Element inserted first in the queue is the first one to be removed.
- Elements are inserted from the rear end and removed from the Front end.
- Queue has two ends **Front** and **Rear**.

Stack

- Last In First Out(LIFO List)
- Insertion and deletion operations occur at only one end. The end is often known as **top** of the stack

Program to implement queue using Array

References

- Data Structures using C, Reema Thareja, Oxford
- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.

Data Structures
Module 2.3
Stack and Queues

Ms. Sarika Dharangaonkar
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Introduction to Circular Queue
- Algorithms for Circular Queue operations
- Implementation of Circular Queue

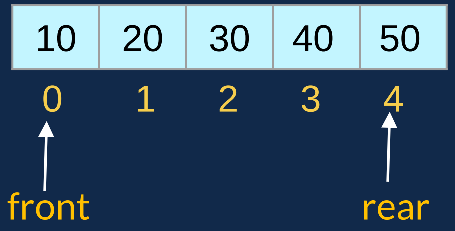
Learning Objectives

At the end of the lecture, students will be able to

- Modify linear queue to work as circular queue.
- Develop program to implement circular queue ADT using array and linked list.

Drawback of Linear Queue

- Queue full condition : rear=SIZE-1
- Suppose Queue is of Size 5 (int queue[5])
- 5 elements are inserted

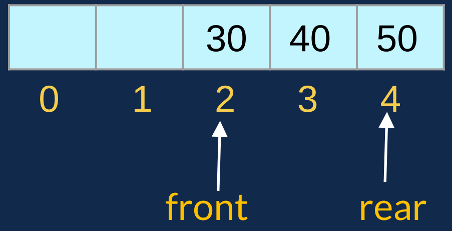


10	20	30	40	50
0	1	2	3	4

front

rear

- 2 elements are deleted



		30	40	50
0	1	2	3	4

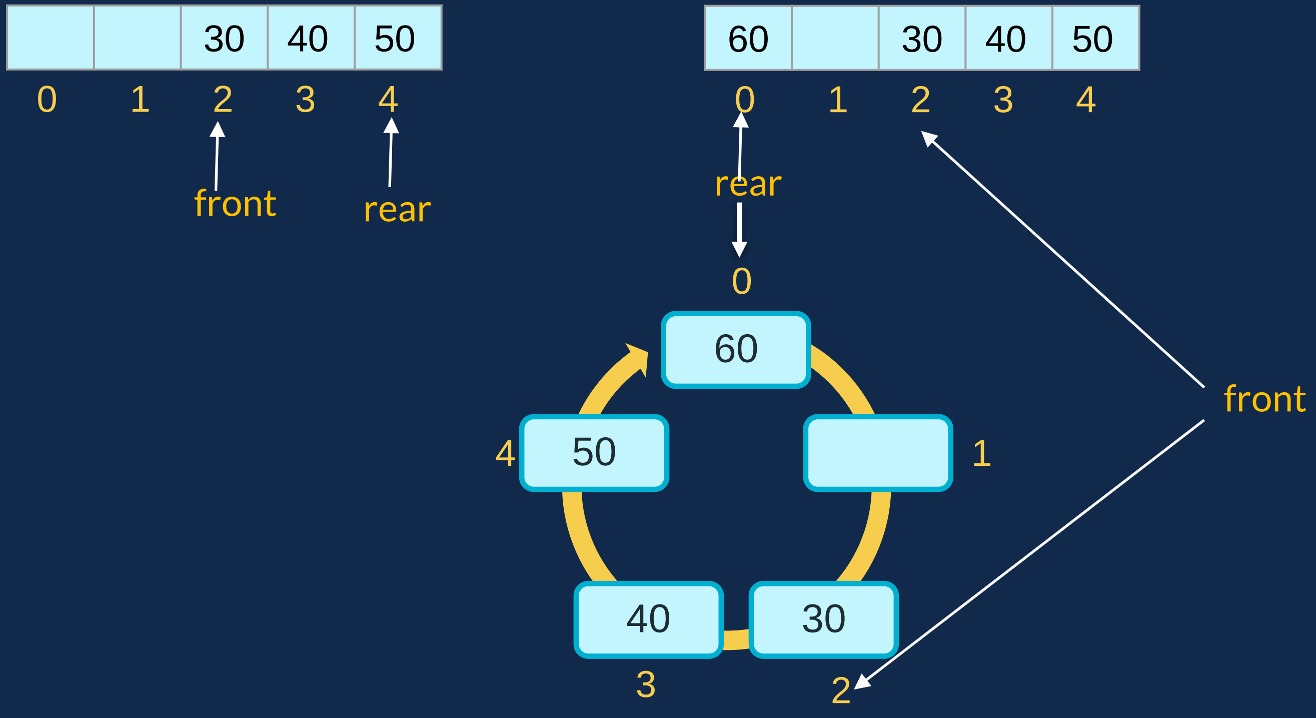
front

rear

- We cannot insert the next element even if there is a space in front of the **queue**.

Circular Queue

Benefit of the **circular queue** is that we can make **use** of the spaces in front of the **queue**.

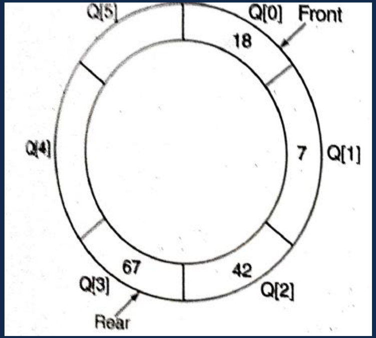


0	1	2	3	4
		30	40	50

0	1	2	3	4
60		30	40	50

Circular queue operations

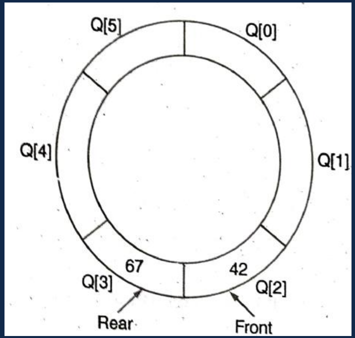
Circular queue after inserting 18, 7, 42, 67



18	7	42	67		
0	1	2	3	4	5

front rear

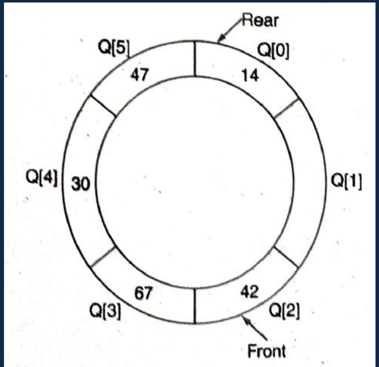
Circular queue after removing 18, 7



		42	67		
0	1	2	3	4	5

front rear

Circular queue after inserting 30, 47, 14

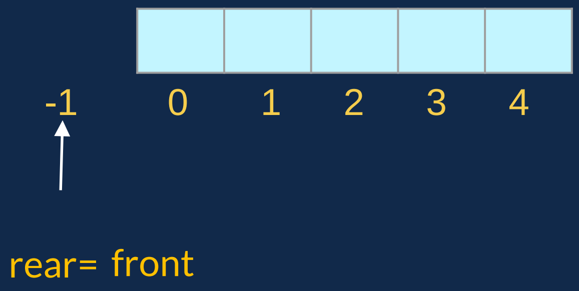


14		42	67	30	45
0	1	2	3	4	5

rear front

Circular Queue initialization

```
#define SIZE 5
int queue[SIZE];
int count=0;
int front=-1;
int rear=-1;
```



Circular queue Empty condition

```
If(count==0)
```

Circular queue Full condition

```
If(count==SIZE)
```

Enqueue Operations on a Circular Queue

Algorithm: enqueue operation

```
Step 1: IF count = SIZE then
            PRINT OVERFLOW & STOP
Step 2:  IF Front=-1 and rear=-1
            SET Front=rear=0
Step 3: SET rear = (rear + 1) % SIZE
Step 4: SET queue[rear] = data
Step 5: SET count = count + 1
 Step 6: END
```

Dequeue Operation on a Circular Queue

Algorithm: dequeue operation

```
Step 1: IF count = 0
            PRINT UNDERFLOW & STOP

Step 2: SET data = queue[front]

Step 3: SET front= (front + 1) % SIZE

Step 4: SET count = count - 1

Step 5: RETURN data
```

Program to implement circular queue using Array

References

- Data Structures using C, Reema Thareja, Oxford
- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.

Data Structures
Module 2.3
Stack and Queues

Ms. Sarika Dharangaonkar,
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Priority Queue
- Introduction of Double Ended Queue
- Applications of Queue.

Learning Objectives

At the end of the lecture, students will be able to

- Implement different types of queues

Priority Queue

- A priority queue is another type of queue structure in which elements can be inserted or delete based on the priority.
- A priority queue is a data structure in which each element has been assigned a value called the priority of the element and an element can be inserted or deleted not only at the ends but at any position on the queue.

Priority Queue

A priority queue is a collection of elements such that each element has been assigned explicit or implicit priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any element of lower priority
2. Two elements with the same priority are processed to the order in which they were inserted to the queue.

Types of Priority Queues

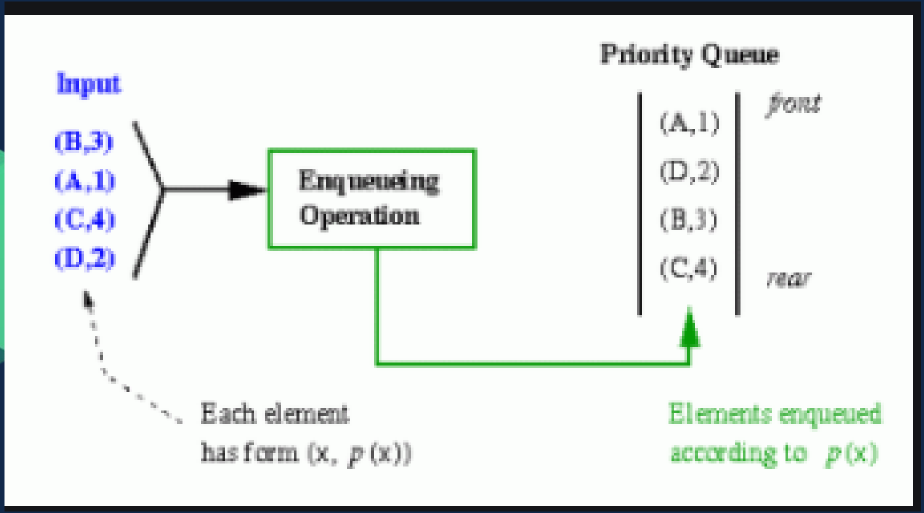
- Ascending priority queue

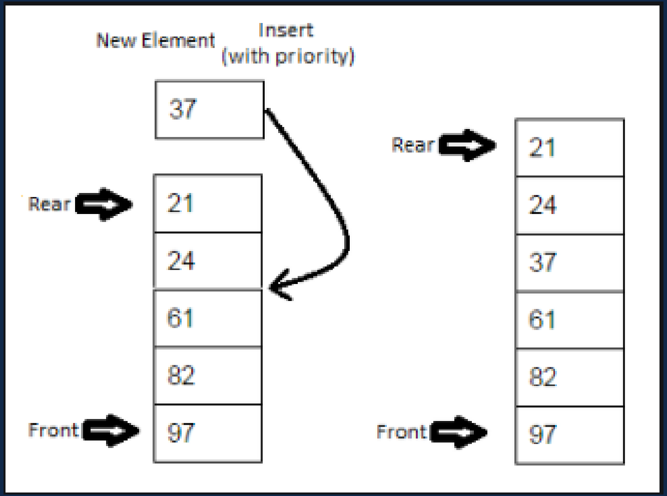
In ascending priority queue elements can be inserted in any order. But, while deleting elements from the queue, always a small element only to be deleted first

- Descending priority queue.

In descending priority queue elements are inserted in any order but while deleting elements from the queue always a largest element is deleted first.

Priority Queues

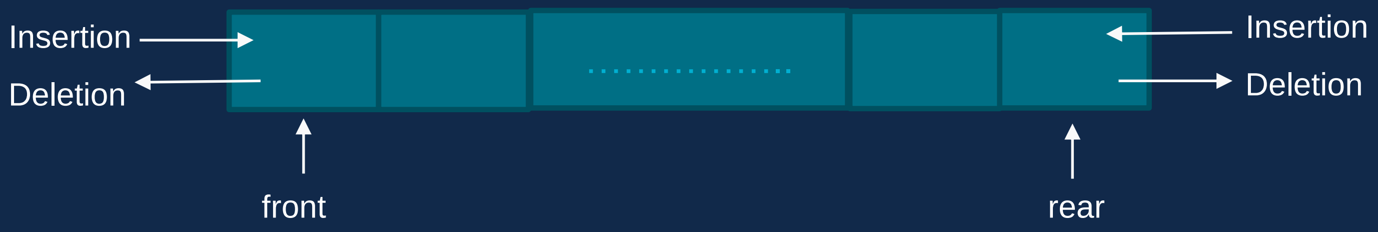




https://youtu.be/1KRbIVZZa48

Double-Ended Queue(DEQUE)

- In Deque, both insertion and deletion operations are performed at either end of the queues.
- We can insert or delete element from the rear end or the front end.



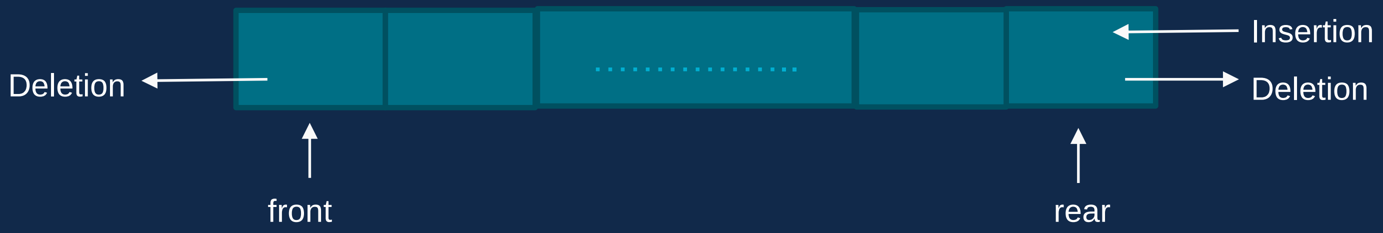
- This deque can be used both as stack and as queue.

Types of Deques

- Input-restricted deque
- Output-restricted deque

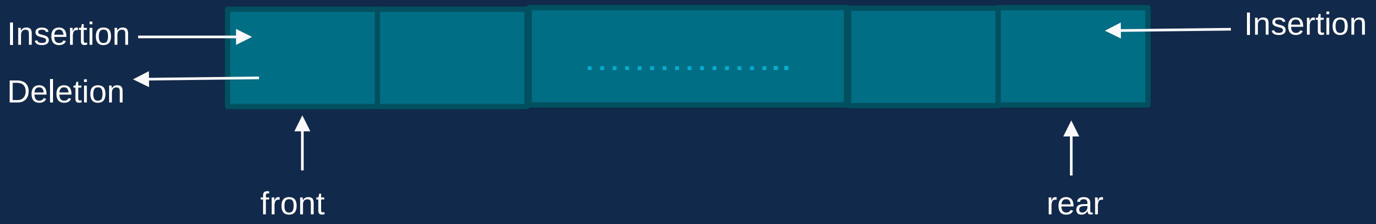
Input-restricted deque

In Input-restricted deque, element can be added at only one end but we can delete the element from both ends.

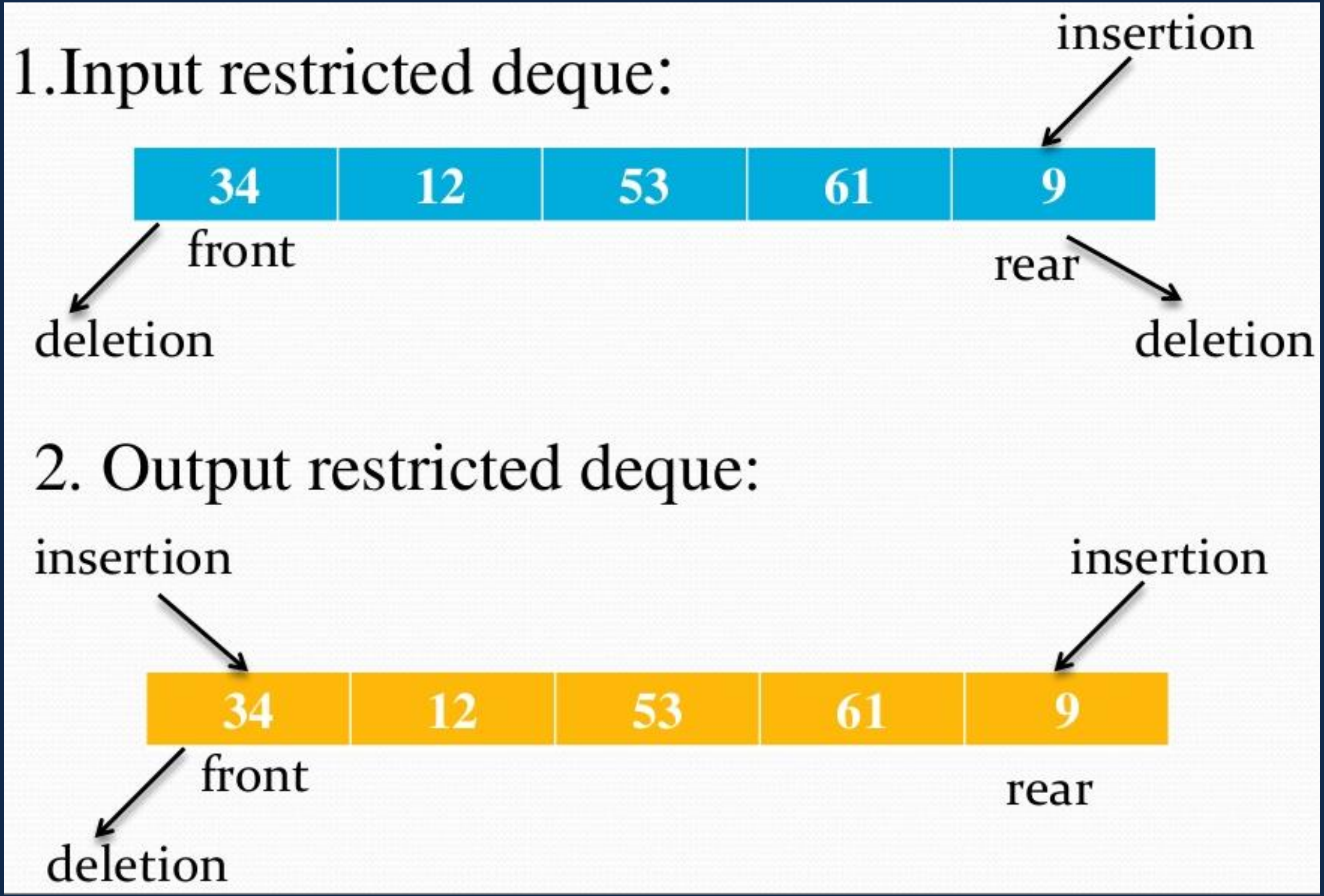


Output-restricted deque

An output-restricted deque is a deque where deletions take place at only one end but allows insertion at both ends.



Example of Deques



1.Input restricted deque:

insertion

34	12	53	61	9

front

rear

deletion

deletion

2. Output restricted deque:

insertion

insertion

34	12	53	61	9

front

rear

deletion

Condition for insertion and deletion in Deque

- When an attempt is made to insert an element into a deque which is already full, overflow occurs.

- When an attempt is made to delete an element from a deque which is empty, underflow occurs.

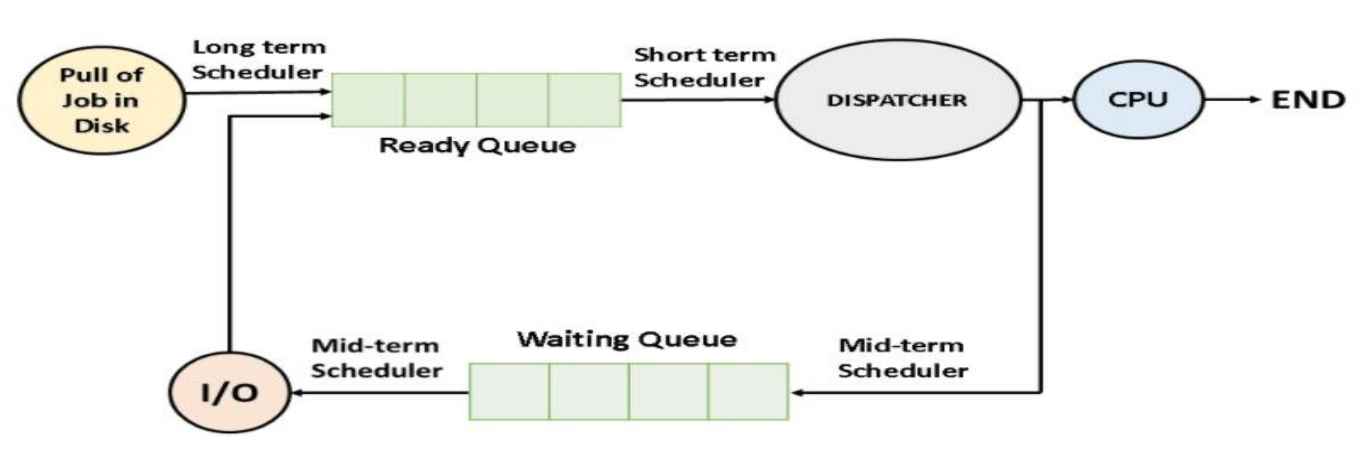
Deque operations

- Add an element at the rear end
- Add an element at the front end.
- Delete an element from the front end.
- Delete an element from the rear end.

Application of Queue

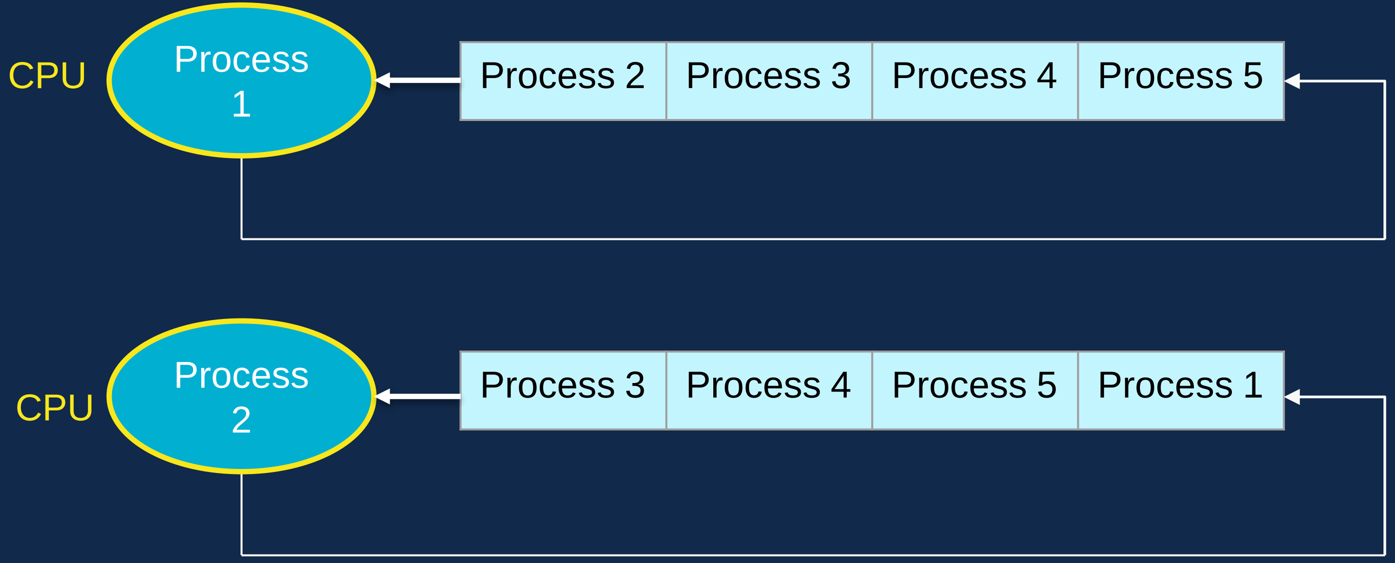
CPU Scheduler

In a multitasking operating system, the CPU time is shared between multiple processes. At a given time, only one process is running, all the others are ‘waiting’. The CPU time is administered by the scheduler. The scheduler keeps all current processes in a queue with the active process at the front of the queue.



Round-Robin scheduling

Every process is granted a specific amount of CPU time, its ‘quantum’. If the process is still running after its quantum run out, it is suspended and put towards the end of the queue.

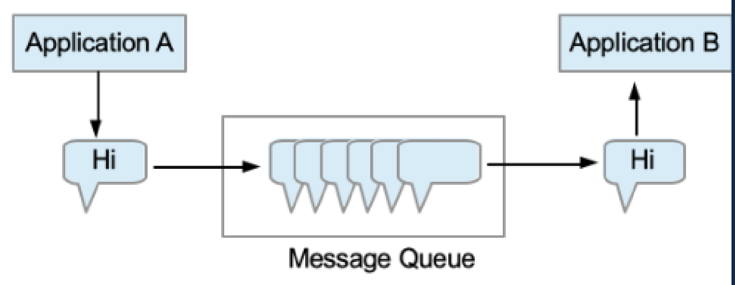


Serving Requests

In computing, it often happens that many processes require the service of a single, shared resource. For example, usually several people share the use of a printer. If requests for printing are submitted faster than they can be satisfied by the printer, the requests are placed on a queue so as to preserve the order in which the requests were submitted. New requests are added at the end of the queue, and, when the printer finishes one request it starts on the request at the front of the queue. Another computing resource that is usually shared by many users is mass-storage devices, such as large disks

Buffers

The other main use of queues is storing data that is being transferred asynchronously between two processes. Asynchronous means that the sender and receiver are not synchronized i.e. that the receiver is not necessarily ready/able to receive the data at the time and speed at which the sender is sending it. A queue is placed between the two processes: the sender enqueues the data whenever it likes, the receiver serves out the data whenever it likes. Such a queue is sometimes called a buffer.



Types of Expression

Infix - 4 + 5 * 5

The default way of representing a mathematical expression is in the form of Infix notation. This method is called Polish Notation (discovered by the Polish mathematician Jan Lukasiewicz).

Prefix - + 4 * 5 5 (The Operators precede the operands.)

Postfix - 4 5 5 * + (The Operators follow the operands.)

The Valuable aspect of RPN (Reverse Polish Notation or Postfix)

- Parentheses are not necessary
- Easy for a compiler to evaluate an arithmetic expression.

Postfix (Reverse Polish Notation)

Postfix notation arises from the concept of post-order traversal of an expression tree. For now, consider postfix notation as a way of redistributing operators in an expression so that their operation is delayed until the compile time.

References

- Data Structures using C, Reema Thareja, Oxford

- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.