Data Structures (CSE2PM01A / AID2PM01A)

S. Y. B. Tech CSE / AIDS   **Semester – III**

DEPARTMENT OF COMPUTER ENGINEERING AND TECHNOLOGY

Syllabus

Unit 3 :

Stacks: Stack as an Abstract Data Type, Representation of Stack Using Sequential Organization, stack operations, Multiple Stacks Applications of Stack- Expression Conversion and Evaluation, Linked Stack and Operations, Recursion.

Queues: Queue as Abstract Data Type, Representation of Queue Using Sequential Organization, Queue Operations Circular Queue, Advantages of Circular queues, Linked Queue and Operations Deque-Basic concept, types (Input restricted and Output restricted), Application of Queue: Job scheduling.

Queue

Topics to be Covered

❑ Queue as Abstract Data Type

❑ Representation of Queue Using Sequential Organization

❑ Queue Operations

❑ Circular Queue, Advantages of Circular queues

❑ Linked Queue and Operations

❑ Deque-Basic concept, types (Input restricted and Output restricted)
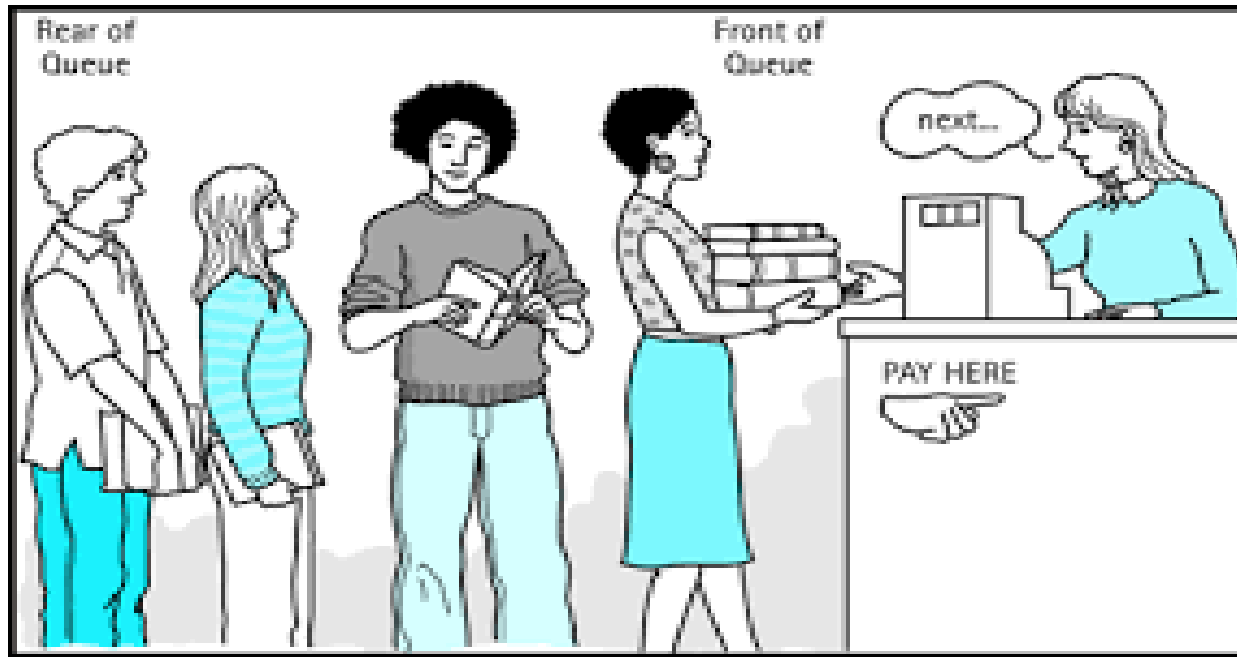
❑ Application of Queue: Job scheduling.
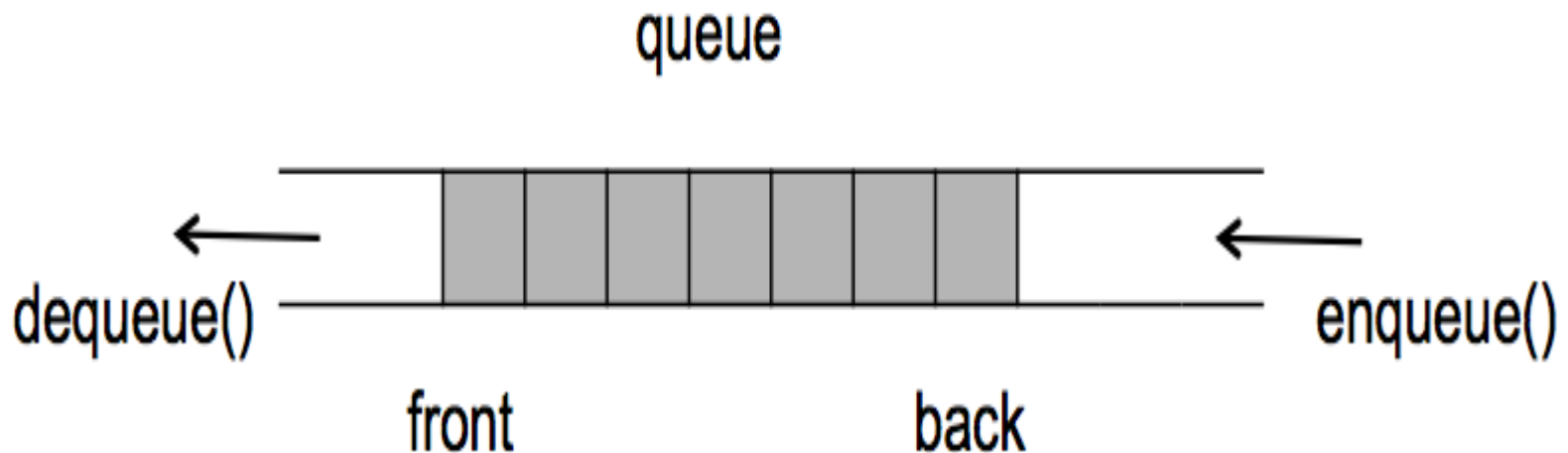
Unit : III

Queue

QUEUE

Queue

Queue as a Data Structure

Queue

Queue is an ordered list (linear data structure) in which insertions(Enqueue) are done at rear end and deletions(dequeue) are done at the front end of the Queue.

queue

dequeue()   enqueue()

front   back

ADT of Queue

CREATEQ(*Q) which creates Q as an empty queue;*

ADDQ(*i,Q) which adds the element i to the rear of a queue and returns the new queue;*

DELETEQ(*Q) which removes the front element from the queue Q and returns the resulting queue;*

FRONT(*Q) which returns the front element of Q;*

ISEMTQ(*Q) which returns true if Q is empty else false.*

ADT of Queue (Cont')

A complete specification of this data structure is

---

structure *QUEUE (item)*

    declare **CREATEQ( ) queue**

        $ADDQ(item, queue) \rightarrow queue$

        $DELETEQ(queue) \rightarrow queue$

        $FRONT(queue) \rightarrow item$

        $ISEMTQ(queue) \rightarrow boolean;$

  for all **Q** ε **queue, i** ε **item let**

        $ISEMTQ(CREATEQ) :: =$ **true**

        $ISEMTQ(ADDQ(i, Q)) :: =$ **false**

        $DELETEQ(CREATEQ) :: = error$

        $DELETEQ(ADDQ(i,Q)):: =$ **if ISEMTQ(Q) then CREATEQ**

                                  else $ADDQ(i, DELETEQ(Q))$

         $FRONT(CREATEQ) :: = error$

         $FRONT(ADDQ(i,Q)) :: =$ **if ISEMTQ(Q) then i else FRONT(Q)**

  end

end *QUEUE*

Applications of Queue

Queues, like stacks, also arise quite naturally in the computer solution of many problems.

The most common occurrence of a queue in computer applications is for the scheduling of jobs.

In batch processing the jobs are "queued-up" as they are read-in and executed, one after another in the order they were received.

Operations on Queue

Adding an element at the rear of the Queue

Delete the front element from the queue

PeepRear returns the rear element of the queue

PeepFront returns the front element of the queue

isFull returns if queue is full

isEmpty returns if queue is empty

Adding a element

Algorithm AddQ(q[],elem)

{

    if(isfull())

      print "Queue is full "

   else

    { rear=rear+1

     q[rear]=elem

    }

}

int isfull()

{   if(rear==size-1)

     return 1

  else

    return 0

}

Deleting an element from Queue

Algorithm DelQ(q[])

{

    if(isempty())

        return -1

  else

    {   front=front+1

        elem=q[front]

        return elem

    }

}

int isempty()

{   if(rear==front)

        return 1

  else

        return 0

}

Queue operations

AddQ(10)

AddQ(18)

E=delQ()

AddQ(20)

E=delQ()

E=delQ()

E=delQ()

Circular Queue

A more efficient queue representation is obtained by regarding the array *Q(1:n) as circular. It now* becomes more convenient to declare the array as *Q(0:n - 1).*

When rear = n - 1, the next element is entered at *Q(0) in case that spot is free. Using the same conventions as before, front will always point* one position counterclockwise from the first element in the queue.

Circular Queue
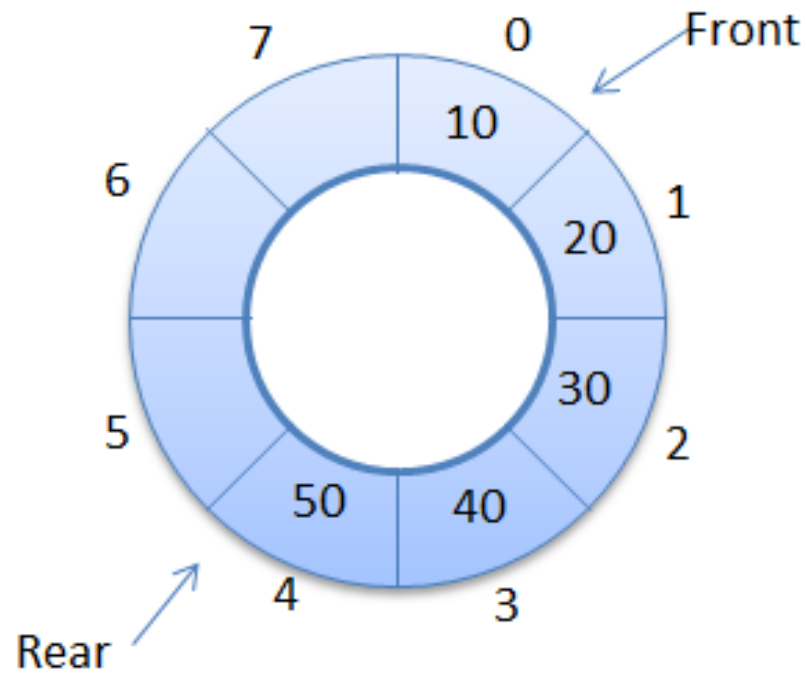
Circular Queue

Initially front=rear=0

Algorithm AddCQ(elem)

{ //insert items in the CQ stored in Q[0..n-1]

   //rear points to the last item & front is one

   //position counter clockwise from the first

    if (rear +1) %n== front

         print "queue full"

      else

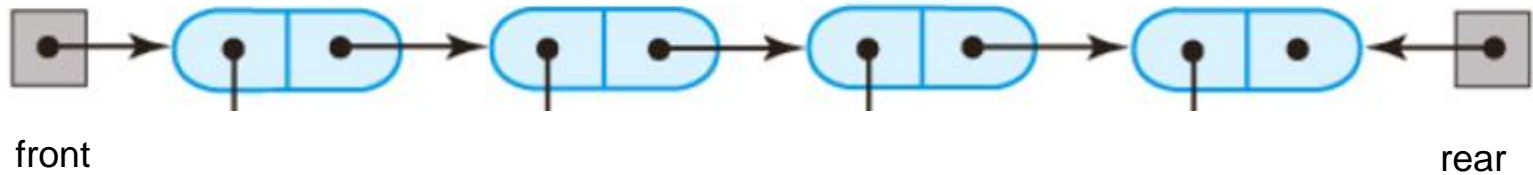   {   rear=(rear+1) %n

         Q[rear]=elem

       }

}

Circular Queue

Algorithm DelCQ(elem)

{ //removes the front element of the queue

  if front==rear

    print "queue empty"

  else

  { front=(front+1) %n

    elem=Q[front]

    return elem

  }

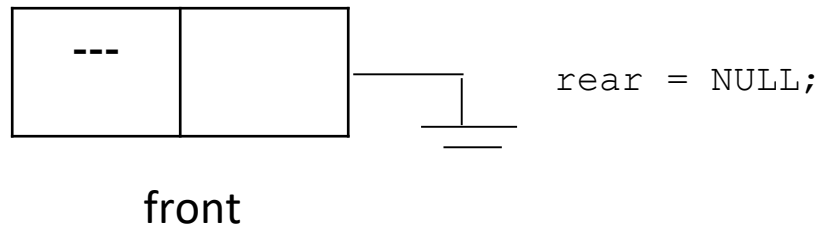}

Linked Queue and Operations

Queue implemented using Singly linked list

Linked Queue and Operations

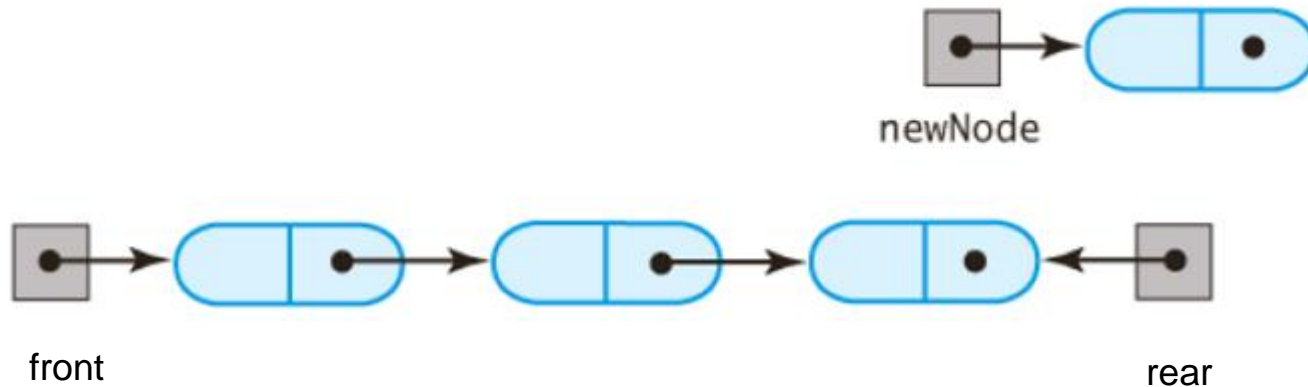Empty queue will have front and rear with the following conditions:
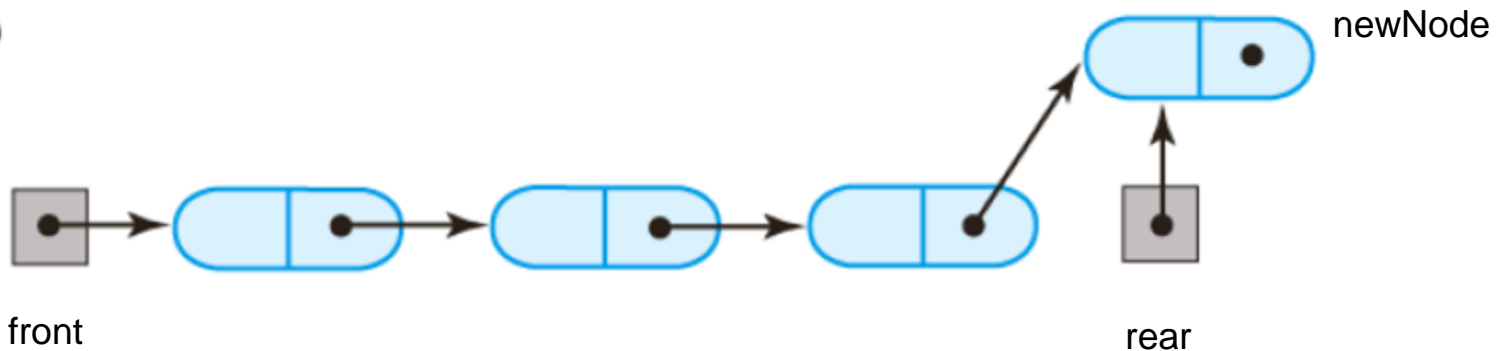
front->next=NULL
rear=NULL

rear = NULL;

front

Linked Queue and Operations
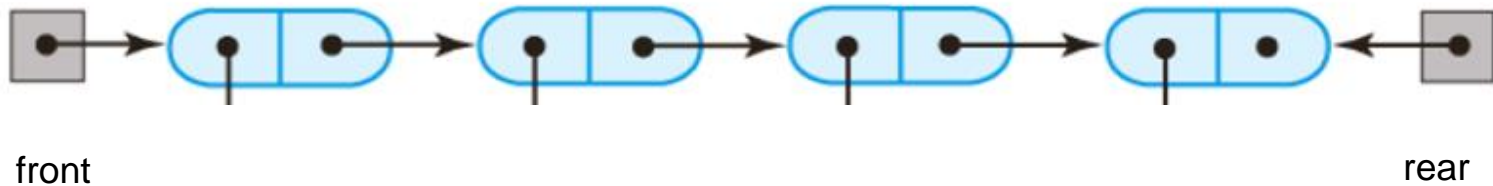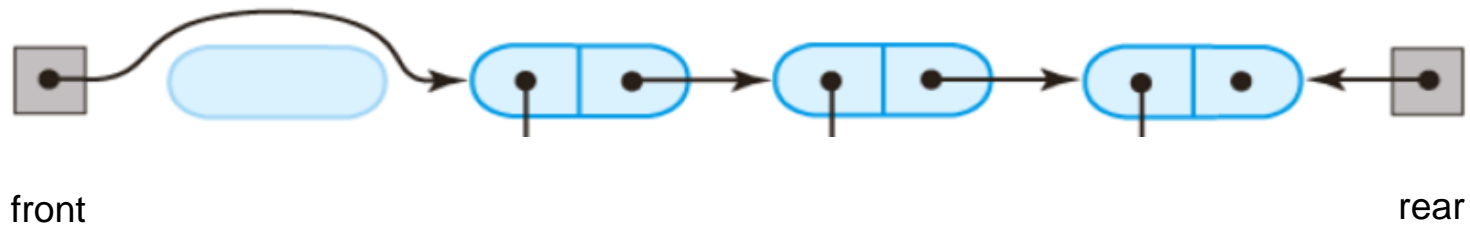

Enqueue operation: a) New node created b) New node inserted (Enqueue complete)

Linked Queue and Operations

(a)

front   rear

(b)

front   rear

Dequeue operation: a) Before deletion b) First node deleted (Dequeue complete)

Linked Queue and Operations

struct node

{

    int data;
    struct node *next;

};

struct node *front, *rear;

//following statements to be initialized in main

front=(struct node *)
   malloc(sizeof(struct node));

front->next=NULL;

rear=NULL;

```
int isempty()
{
        if(front->next==NULL)
                return 1;
        else
                return 0;
}
```

Linked Queue and Operations

```
void enqueue(int element )
{
  //allocate memory to new_node
        new_node->data=element;
        new_node->next=NULL;
        if(isempty())
        {
                front->next=rear=new_node;
        }
        else
        {
                rear->next=new_node;
                rear=new_node;
        }
}
```

Linked Queue and Operations

```
int dequeue()
{
        if(isempty())
        {
                //print queue is empty
                return -1;
        }
        else
         {
                temp = front->next;
                value = temp->data;
                front->next=temp->next;
                temp->next=NULL;
                free(temp);
                return (value);
         }
}
```
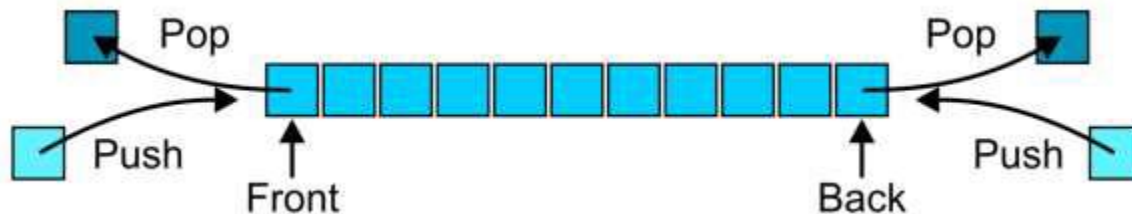
Double Ended Queue(Deque)

- A double-ended queue (deque) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front or rear.

- Hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.

Types of deque

- Input-restricted deque

Deletion can be made from both ends , but Insertion can be made at one end only.

- Output-restricted deque

Insertion can be made at both ends , but Deletion can be made from one end only.

deque Operations

- pushRear() - Insert element at back

- pushFront() - Insert element at front

- popRear() - Remove last element

- popFront() - Remove first element

- isEmpty() – Checks whether the queue is empty or not.

deque Example

| Operation	Deque content
pushFront('a')	['a']
pushFront('b')	['b' , 'a']
pushRear('c')	['b' , 'a' , 'c']
popFront()	['a' , 'c']
popRear()	['a']

Queue Applications: Job Scheduling

❖ Job scheduling is the process of allocating system resources to many different tasks by an operating system (OS).

❖ The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job.

❖ This type of scheduling makes sure that all jobs are carried out fairly and on time.

❖ Job scheduling is performed using job schedulers. Job schedulers are programs that enable scheduling and, at times, track computer "batch" jobs, or units of work

The Operating System maintains the following important process scheduling queues −

Job queue − This queue keeps all the processes in the system.

Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.
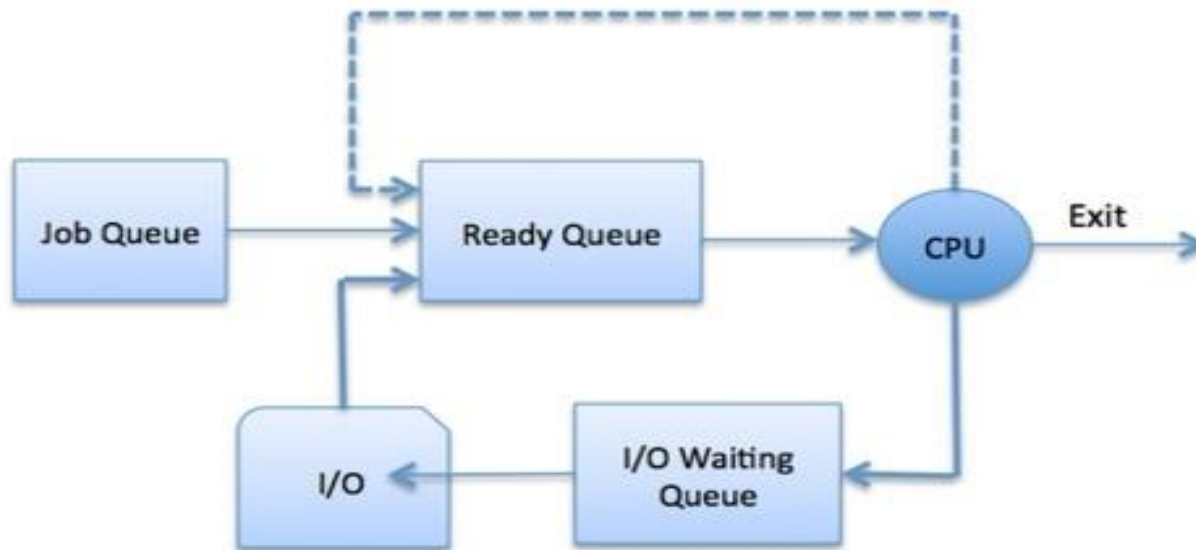
FAQS

1. Write an ADT for Queue
2. What are the primitive operations of Queue.
3. Explain with example Queue applications.

Takeaways

❑ Queue is First in First Out(FIFO) Data Structure

❑ Primitive operations on Queue are enqueue, dequeue,isEmpty and isFull

❑ Queue can be represented by using Array as well as linked list.

❑ Queue is commonly used in Job Sequencing by OS.