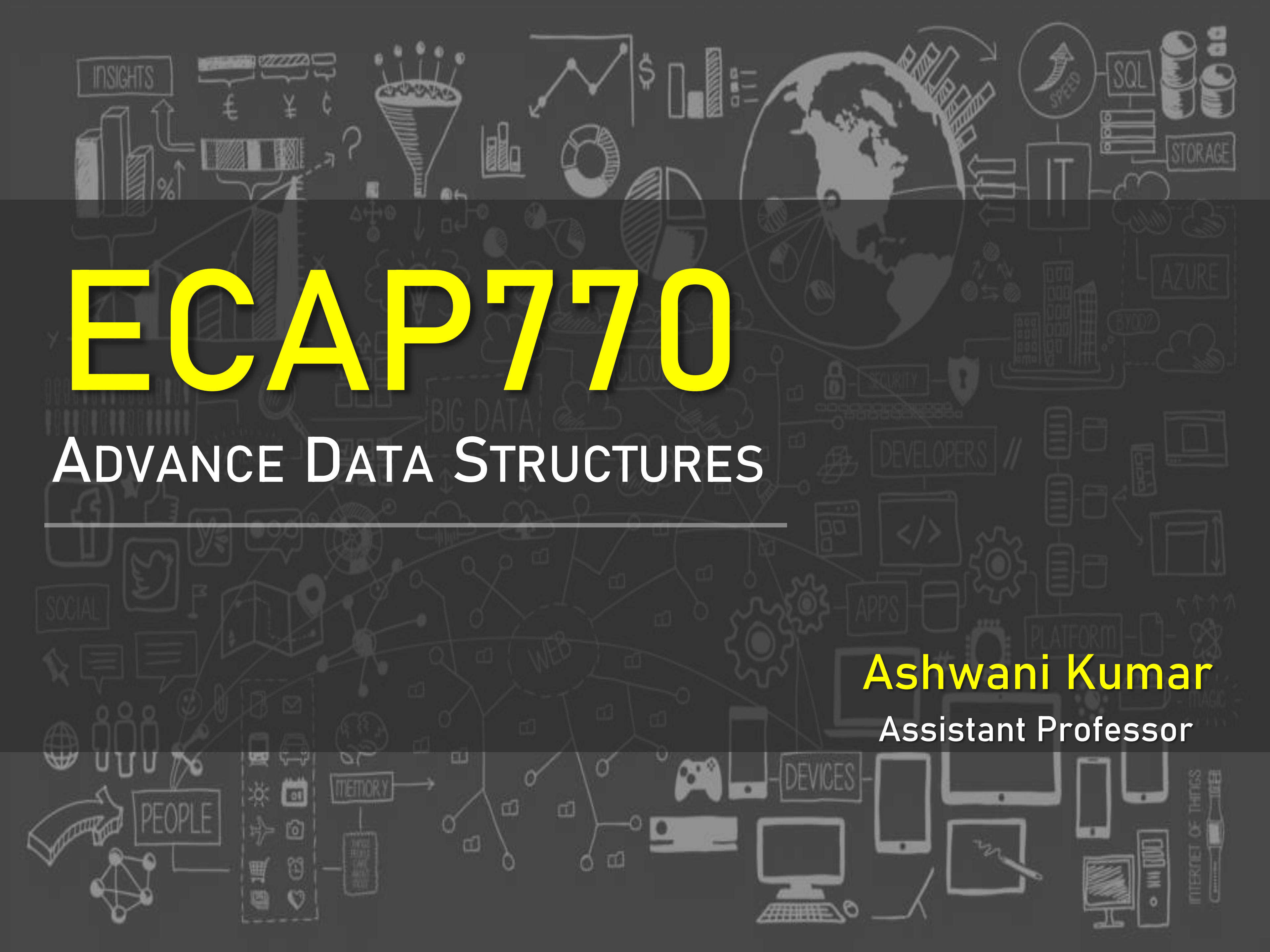


ECAP770

Advance Data Structures

Ashwani Kumar
Assistant Professor

Learning Outcomes

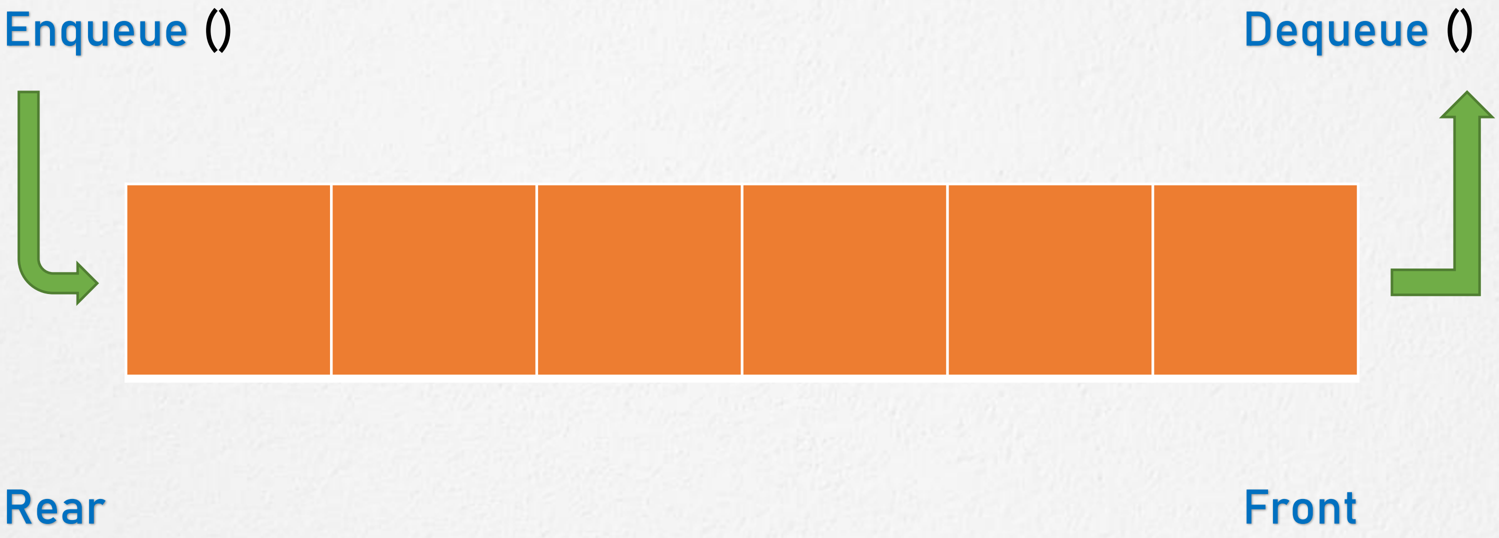
After this lecture, you will be able to

- Understand queue data structure

Queue

- Queue is a linear structure and also called abstract data structure
- Queue follow First In First Out (FIFO) method
- It enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.

Queue representation



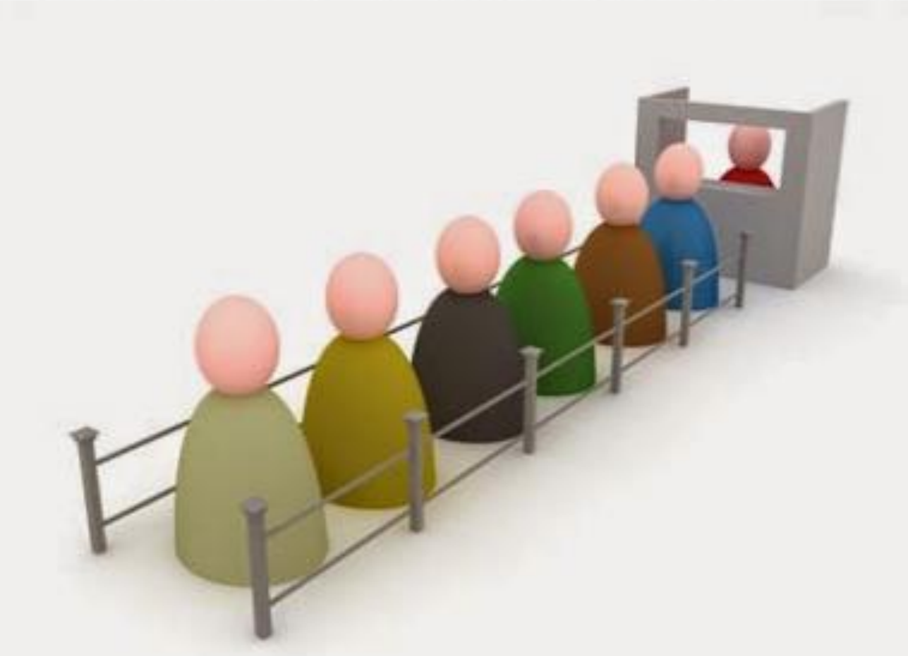
Enqueue ()

Dequeue ()

Rear

Front

Queue



Working of Queue

Queue operations work as follows:

- Two pointers *FRONT* and *REAR*
- *FRONT* track the first element of the queue
- *REAR* track the last element of the queue
- initially, set value of *FRONT* and *REAR* to *-1*

Operations of Queue

- Enqueue: Add an element to the end of the queue
- Dequeue: Remove an element from the front of the queue
- IsEmpty: Check if the queue is empty
- IsFull: Check if the queue is full
- Peek: Get the value of the front of the queue without removing it

Enqueue Operation

- In queue we need to maintain two data pointers, front and rear. Operations on queue are comparatively difficult to implement than that of stacks
- Step 1 – Check if the queue is full.
- Step 2 – If the queue is full, produce *overflow* error and exit.
- Step 3 – If the queue is not full, increment rear pointer to point the next empty space.
- Step 4 – Add data element to the queue location, where the rear is pointing.
- Step 5 – return success.

Algorithm: Enqueue operation

```
procedure enqueue(data)

    if queue is full

    return overflow

  endif

    rear ← rear + 1

  queue[rear] ← data

  return true

  end procedure
```

Implementation of enqueue ()

```
int enqueue(int data)
   if(isfull())
      return 0;
      rear = rear + 1;
   queue[rear] = data;
      return 1;
end procedure
```

Dequeue Operation

- Dequeue operation include two tasks: access the data where front is pointing and remove the data after access.
- Step 1 – Check if the queue is empty.
- Step 2 – If the queue is empty, produce underflow error and exit.
- Step 3 – If the queue is not empty, access the data where front is pointing.
- Step 4 – Increment front pointer to point to the next available data element.
- Step 5 – Return success.

Algorithm: Dequeue operation

```
procedure dequeue
    if queue is empty
    return underflow
  end if
  data = queue[front]
  front ← front + 1
  return true
end procedure
```

Implementation of dequeue()

```
int dequeue() {
    if(isempty())
        return 0;
    int data = queue[front];
    front = front + 1;
    return data;
}
```

Algorithm: isfull()

```
begin procedure isfull

   if rear equals to MAXSIZE

      return true

   else

      return false

   endif

   end procedure
```

Implementation of isfull()

```
bool isfull() {
    if(rear == MAXSIZE - 1)
        return true;
    else
        return false;
}
```

Algorithm: isempty()

```
begin procedure isempty
   if front is less than MIN  OR front is greater than rear
      return true
   else
      return false
   endif
   end procedure
```

Implementation of isempty()

```
bool isempty() {
    if(front < 0 || front > rear)
        return true;
    else
        return false;
}
```

Queue implementation

- Queue can be implemented using:
- Array
- Stack
- Linked List

Applications of Queue

In Operating systems:

a) Semaphores

b) FCFS (first come first serve) scheduling,

c) Spooling in printers

d) Buffer for devices like keyboard

In Networks:

a) Queues in routers/ switches

b) Mail Queues

Applications of Queue

- Queues are used in operating systems for handling interrupts.
- Queues are used as buffers in most of the applications like MP3 media player, CD player, etc
- When a resource is shared among multiple consumers.

  CPU scheduling,

  Disk Scheduling.

Types of Queues

- Simple Queue
  - In a simple queue, insertion takes place at the rear and removal occurs at the front. It follows the FIFO (First in First out) rule.
- Circular Queue
  - In a circular queue, the last element points to the first element making a circular link.

Types of Queues

- Priority Queue
  - A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority.
- Double Ended Queue
  - In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. It does not follow the FIFO (First In First Out) rule.

That’s all for now…