Data Structures & Algorithms

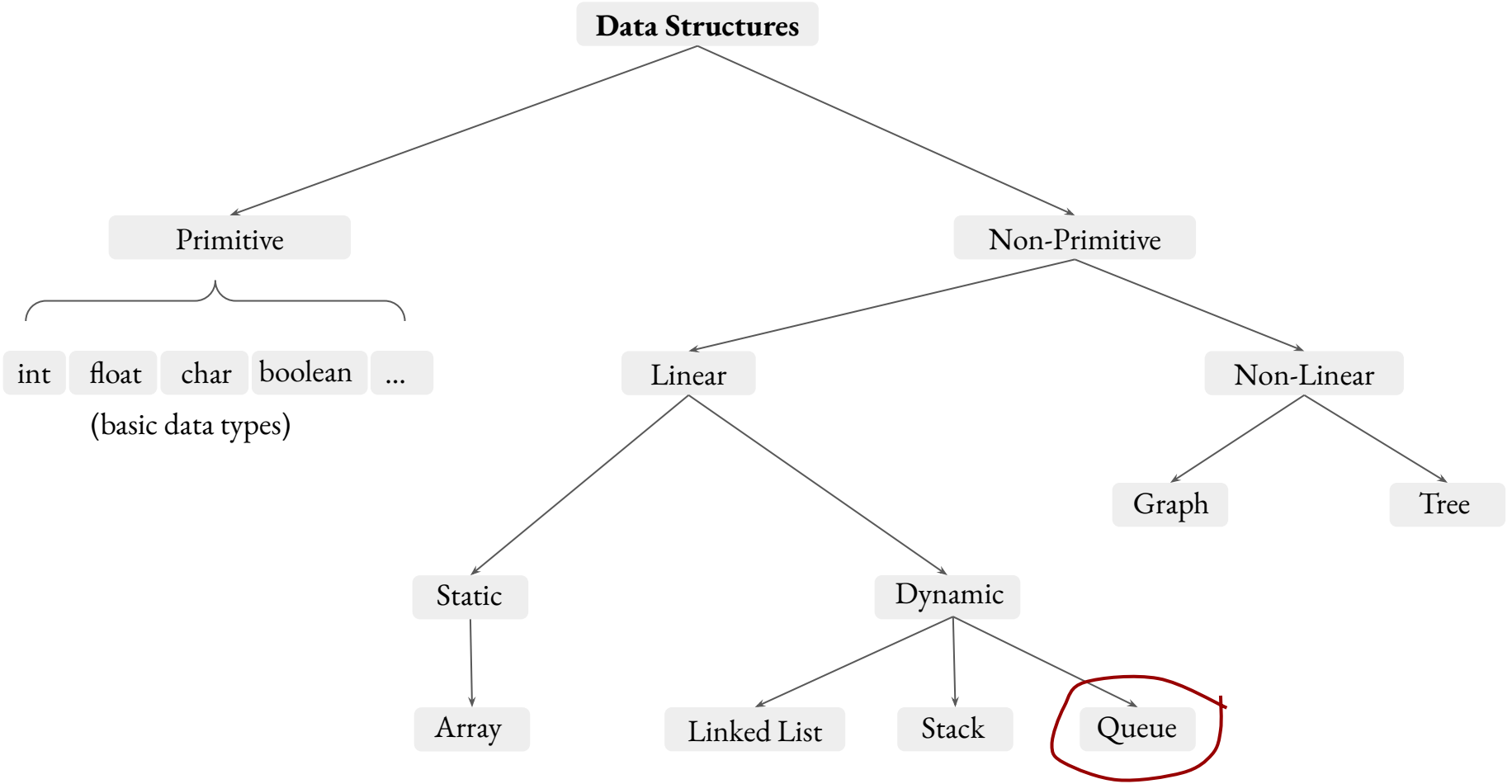
03: Queues

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

Queues
(FIFO: First In First Out)


Data Structures
Primitive
Non-Primitive
int
float
char
boolean
...
(basic data types)
Linear
Non-Linear
Graph
Tree
Static
Dynamic
Array
Linked List
Stack
Queue

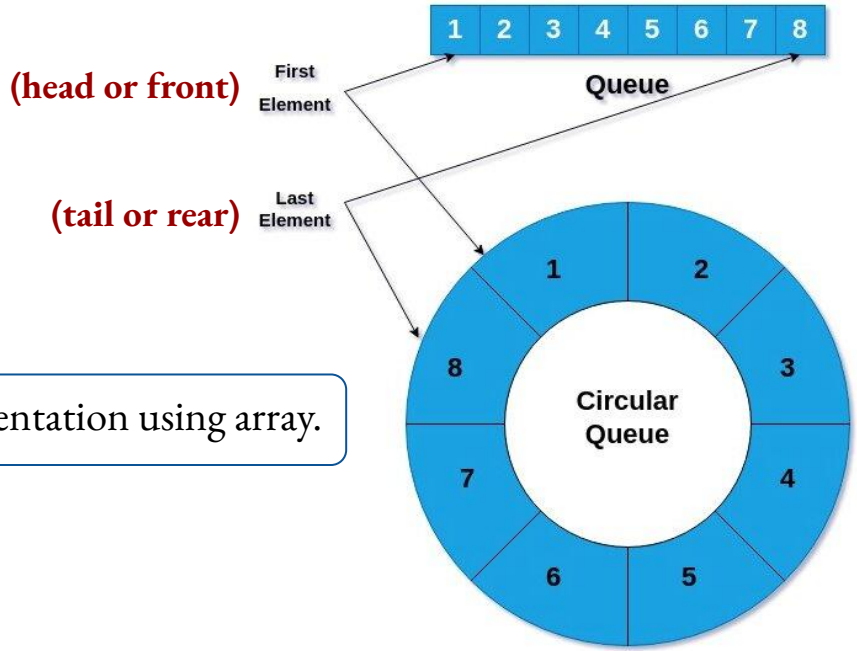
Queues

Queue: A dynamic dataset in which the element deleted is the one that has been in the set for the longest time.

- Queues use a first-in, first-out, or **FIFO**, policy.

Enqueue → Insert Operation

Dequeue → Delete Operation



Circular Queue is a memory efficient implementation using array.

Queues

- Inserting into a queue is **enqueuing**, and deleting from a queue is **dequeuing**.
- The FIFO property of a queue causes it to operate like a line of customers waiting for service. A queue has a **head** and a **tail**.
- When an element is **enqueued**, it goes to the **tail** of the queue, just as a newly arriving customer takes a place at the end of the line.
- The element **dequeued** is the one at the **head** of the queue, like the customer at the head of the line who has waited the longest.

Queues

Some key points about **queues**:

- **Basic operations**:
  - **Enqueue:** Add an element to the **tail** of the queue.

    Enqueuing a full queue causes **overflow**.
  - **Dequeue:** Remove and return the element at the **head** of the queue.

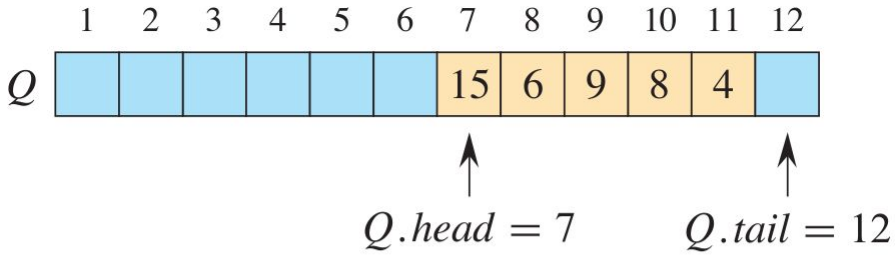
    Dequeuing an empty queue causes **underflow**
  - **Peek:** Examine the element at the front (head) without removing it.
  - **IsEmpty:** Check if the queue is empty.
  - **IsFull:** Check if the queue is full (this depends on the implementation).

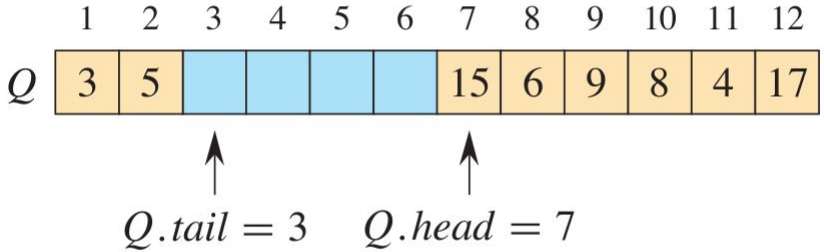
Real-World Examples of Queues

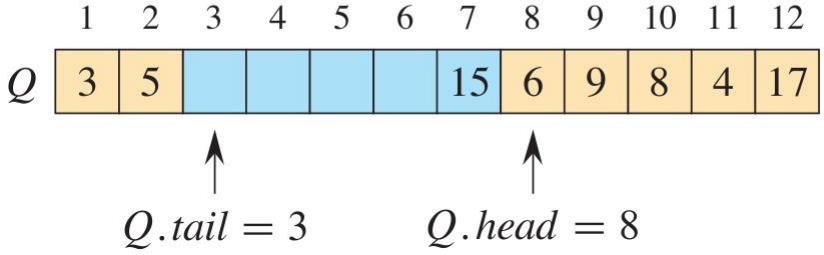
- **Operating Systems:** Operating systems use queues to manage processes and tasks, assigning them resources on a FIFO basis.
- **Print Queue:** When you send multiple documents to a printer, a queue manages the order of their printing.
- **Web Server Requests:** Web servers handle incoming requests in a queue-like fashion to ensure they are processed in order.
- **Customer Service Lines:** Calls waiting to be answered by support agents often form a queue structure.

Pseudocode for Queue operations

Queues

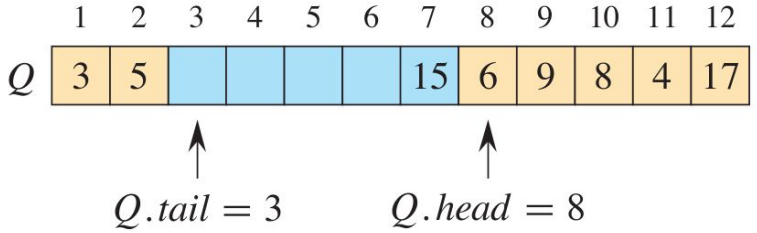

1 2 3 4 5 6 7 8 9 10 11 12
Q
15 6 9 8 4
Q.head = 7
Q.tail = 12


1 2 3 4 5 6 7 8 9 10 11 12
Q
3 5 15 6 9 8 4 17
Q.tail = 3
Q.head = 7


1 2 3 4 5 6 7 8 9 10 11 12
Q
3 5 15 6 9 8 4 17
Q.tail = 3
Q.head = 8

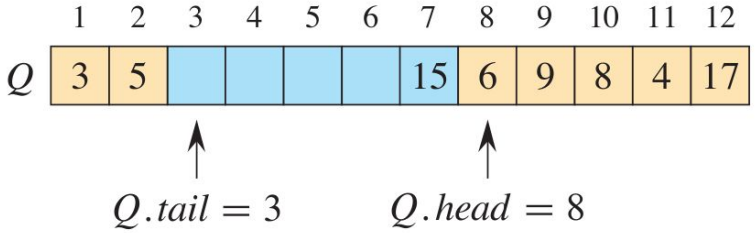
Queue Implementation

Efficient version - Circular Queue



- $Q.head$ indexes the head.
- $Q.tail$ indexes the next location at which a new element will be inserted into the queue.
- Elements reside in $Q.head, Q.head + 1, \ldots, Q.tail - 1$, wrapping around so that $Q[1]$ follows $Q[n]$.
- Initially, $Q.head = Q.tail = 1$.
- $Q.head = Q.tail \Rightarrow$ queue is empty. Attempting to dequeue causes underflow.
- $Q.head = Q.tail + 1$ or both $Q.head = 1$ and $Q.tail = n \Rightarrow$ the queue is full. Attempting to enqueue causes overflow.
- Attribute $Q.size$ gives the size n of the array.

Queues




```
ENQUEUE(Q, x)
  Q[Q.tail] = x
  if Q.tail == Q.size
      Q.tail = 1
  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q, n)
  x = Q[Q.head]
  if Q.head == Q.size
      Q.head = 1
  else Q.head = Q.head + 1
  return x
```

Exercise:

- How to include the error checking for overflow and underflow?
- Implement PEEK(Q) which gives which element is next deleted.

C Program for Queue operations (Array implementation)