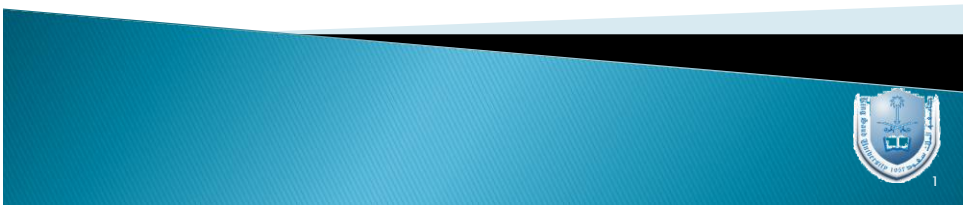
Queue

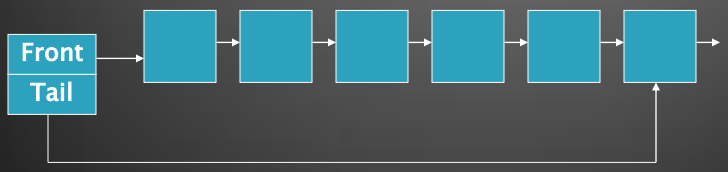
CSC212:Data Structure



Queue

- Queue: First In First Out (FIFO).
  - Used in operating systems, simulations etc.
- Priority Queues: Highest priority item is served first.
  - Used in operating systems, printer servers etc.

Queues



ADT Queue: Specification

Elements: The elements are of a variable type <Type>. In a linked implementation elements are placed in nodes.

```
public class Node<T> extends Object {
  public T data;
  public Node<T> next;
public Node () {
     data = null;  next = null; }
public Node (T val) {
     data = val;  next = null; }
}
```

ADT Queue: Specification

Structure: the elements are linearly arranged, and ordered according to the order of arrival, most recently arrived element is called the tail and least recently arrived element the front or head.

Domain: the number of elements in the queue is bounded therefore the domain is finite. Type of elements: Queue

5

ADT Queue: Specification

Operations:

1. **Method** Enqueue (Type e)
   requires: Queue Q is not full. **input:** Type e.
   results: Element e is added to the queue at its tail. **output:** none.
2. **Method** Serve (Type e)
   requires: Queue Q is not empty. **input**:
   results: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
3. **Method** Length (int length)
   input: **results**: The number of element in the Queue Q is returned. **output**: length.

6

ADT Queue: Specification

Operations:

4. **Method** Full (boolean flag).
   requires: **input**:
   results: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.

ADT Queue (Linked Implementation)

```
public class LinkQueue <Type> {
    private Node<Type> head, tail;
    private int size;

    /** Creates a new instance of LinkQueue */
public LinkQueue() {
        head = tail = null;
        size = 0;
    }
```

ADT Queue (Linked Implementation)

```
public boolean full() {
        return false;
    }
public int length (){
        return size;
    }
```

ADT Queue (Linked Implementation)

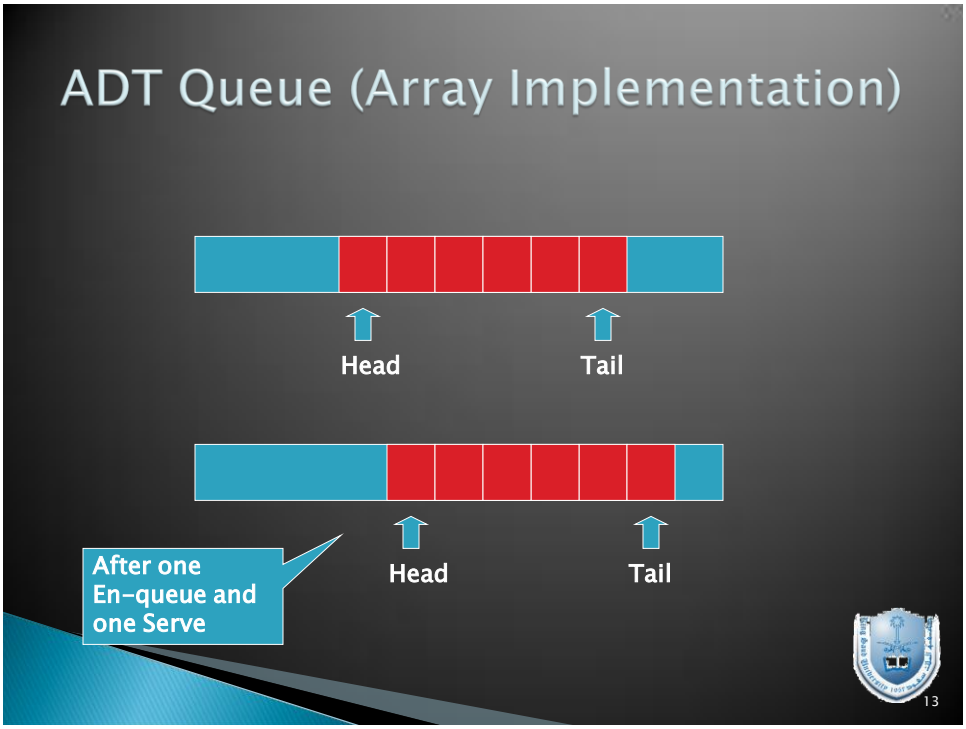
```
public void enqueue (Type e) {
        if (tail == null){
            head = tail = new Node(e);
        }
        else {
            tail.next = new Node(e);
            tail = tail.next;
        }
        size++;
    }
```

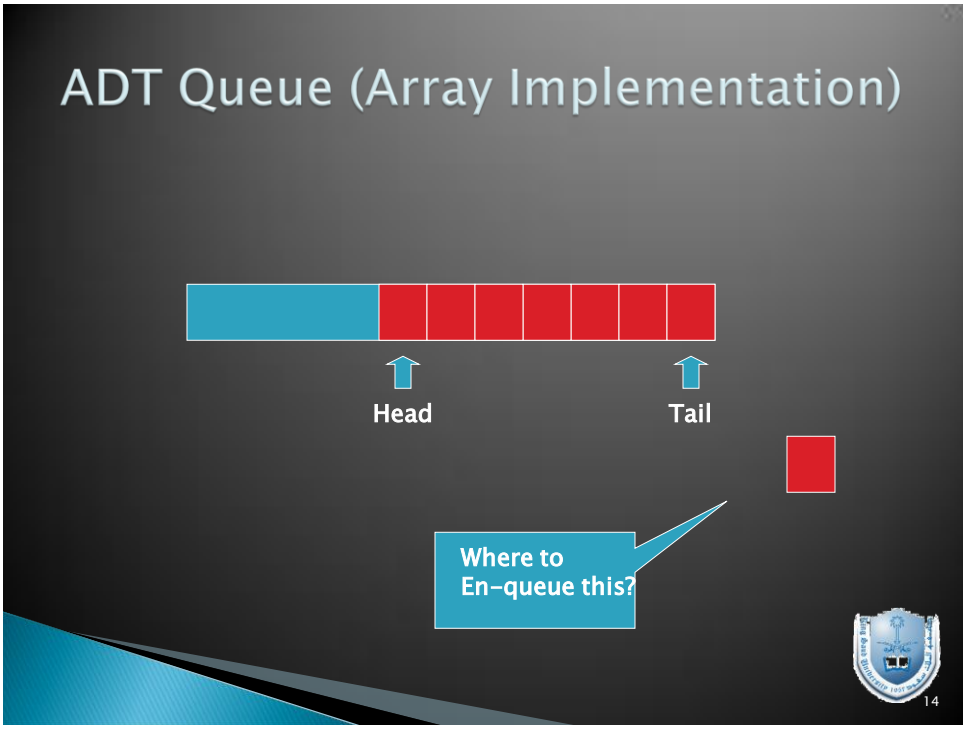
ADT Queue (Linked Implementation)

```
public Type serve() {
        Type x;
        x = head.data;
        head = head.next;
        size--;
        if (size == 0)
            tail = null;
        return x;
    }
}
```

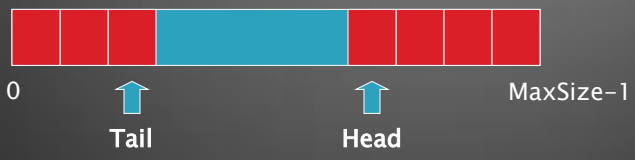
ADT Queue (Array Implementation)

- Array implementation of the queue…a fixed size array is used to store the data elements.
- As data elements are enqueued & served the queue crawls through the array from low to high index values.
- As the queue crawls forward, it also expands and contracts.


ADT Queue (Array Implementation)
Head
Tail
Head
Tail
After one
En-queue and
one Serve
13

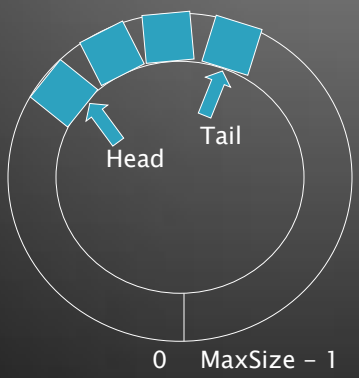

ADT Queue (Array Implementation)
Head
Tail
Where to
En-queue this?
14

ADT Queue (Array Implementation)



Wrap Round

ADT Queue (Array Implementation)



ADT Queue (Array Implementaion)

```
public class ArrayQueue <T> {
    private int maxsize;
    private int size;
    private int head, tail;
    private T[] nodes;
    /** Creates a new instance of ArrayQueue */
public ArrayQueue(int n) {
        maxsize = n;
        size = 0;
        head = tail = 0;
        nodes = (T[]) new Object[n];
    }
```

17

ADT Queue (Array Implementation)

```
public boolean full () {
        return size == maxsize ? true : false;
    }
public int length () {
        return size;
    }
public void enqueue(T e) {
        nodes[tail] = e;
        tail = (tail + 1) % maxsize;
        size++;
    }
```

18

ADT Queue (Array Implementation)

```
public T serve () {
        T e = nodes[head];
        head = (head + 1) % maxsize;
        size--;
        return e;
    }
}
```

Priority Queue

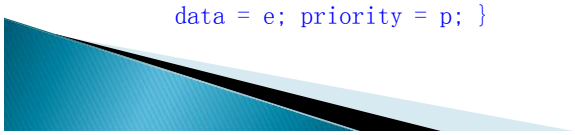
- Each data element has a priority associated with it.Highest priority item is served first.
- Real World Priority Queues: hospital emergency rooms…most sick patients treated first, events in a computer system, etc.
- Priority Queue can be viewed as:
  - View 1: Priority queue as an ordered list.
  - View 2: Priority queue as a set.

ADT Priority Queue

Specification:

Elements: The elements are of type PQNode. Each node has in it a data element of variable type <Type> and priority of type Priority (which could be int type). [To implement encapsulation, You can have private data members with getters and setters]

```
public class PQNode<T> {
    public T data;
    public Priority priority;
    public PQNode<T> next;
    public PQNode() {
        next = null; }
    public PQNode(T e, Priority p) {
        data = e; priority = p; }
```



21

ADT Priority Queue

Structure: the elements are linearly arranged, and may be ordered according to a priority value, highest priority element is called the front or head and least priority element the tail.

Domain: the number of nodes in the queue is bounded therefore the domain is finite. Type of elements: PriorityQueue

22

ADT Priority Queue

Operations:

1. **Method** Enqueue (Type e, Priority p)
   requires: PQ is not full. **input:** e, p.
   results: Element e is added to the queue according to its priority. **output:** none.
2. **Method** Serve (Node<Type> node)
   requires: PQ is not empty. **input**: None
   results: the element at the head of PQ is removed and returned. **output**: node.
3. **Method** Length (int length)
   input: **results**: The number of element in the PQ is returned. **output**: length.

23

ADT Priority Queue

Operations:

4. **Method** Full (boolean flag).
   requires: **input**:
   results: If PQ is full then flag is set to true, otherwise flag is set to false. **output**: flag.

24

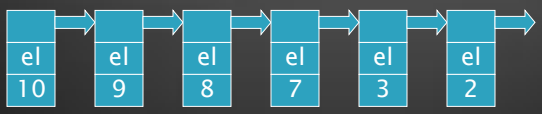
ADT Priority Queue

Array Implementation

Head Tail

Insert Where?

Linked Implementation



Head Tail

25

ADT Priority Queue (Linked)

```
public class LinkPQ<T> {
    private int size;
    private PQNode<T> head, tail;
  /* tail is of no use here. */
    public LinkPQ() {
        head = tail = null;
        size = 0;
    }
    public int length (){
        return size;    }
```

26

ADT Priority Queue (Linked)

```
public int length (){
        return size;
    }
public boolean full () {
        return false;
    }
```

ADT Priority Queue (Linked)

```
public void enqueue(T e, int pty) {
       PQNode<T> p, q, tmp;
       if ((size == 0) || (pty > head.priority)) {
           tmp = new PQNode<T>(e, pty);
           tmp.next = head;
           head = tmp; }
       else {
           p = head; q = null;
           while ((p != null) && (p.priority > pty)) {
               q = p; p = p.next; }
           tmp = new PQNode<T>(e, pty);
           tmp.next = p; q.next = tmp; } }
```

ADT Priority Queue (Linked)

```
public Node<T> serve (){
        Node<T> node = head;
        head = head.next;
        size--;
        return(node);
    }
}
```

ADT Priority Queue

- Implementations
  - Array Implementation: Enqueue is O(n), Serve is O(1).
  - Linked List: Enqueue is O(n), Serve is O(1).
  - Heap: Enqueue is O(log n), Serve is O(log n) ← Heaps to be discussed later.

Double-Ended Queues

- Double ended queue (or a **deque**) supports insertion and deletion at both the front and the tail of the queue.
- Supports operations: addFirst( ), addLast(), removeFirst( ) and removeLast( ).
- Can be used in place of a queue or a stack.

Double-Ended Queues

Operations: (Assume all operations are performed on deque DQ)

1. **Method** addFirst (Type e)
   requires: DQ is not full. **input:** e.
   results: Element e is added to DQ as first element. **output:** none.
2. **Method** addLast (Type e)
   requires: DQ is not full. **input**: e
   results: Element e is added to DQ as last element. **output**: none.
3. **Method** removeFirst (Type e)
   requires: DQ is not empty. **input**: none **results**: Removes and returns the first element of DQ. **output**: e.

Double-Ended Queues

4. **Method** removeLast (Type e)
   requires: DQ is not empty. **input:** none.
   results: Removes and returns the last element of DQ.
   output: e.
5. **Method** getFirst (Type e)
   requires: DQ is not empty. **input**: none
   results: Returns the first element of DQ. **output**: e.
6. **Method** getLast (Type e)
   requires: DQ is not empty. **input**: none
   results: Returns the last element of DQ. **output**: e
7. **Method** size (int x)
   input: none **results**: Returns the number of elements in DQ.
   output: x

33

Double-Ended Queues

8. **Method** isEmpty (boolean x)
   input: none **results**: if DQ is empty returns x as true otherwise false. **output**: x

34

ToDo

- Read 5.2, 5.3 of the Textbook.
- Add “int length()” method in the LinkQueue class with O(n) complexity.
- Add “`int length(ArrayQueue<T> q)` ” in the Test class of ArrayQueue. The Queue must remain unchanged after the operation.
- Implement DQueue (Double ended queue) using a Java class using Link List.
- Test this DQueue using a test Class.