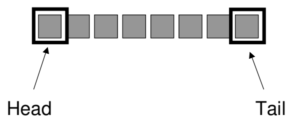
Queues

SE202 Data Structures

Queues

- A queue is another special type of list
  - insertions are made at one end, called the tail of the queue
  - deletions take place at the other end, called the head
  - thus, the last one added is always the last one available for deletion
  - also referred to as
    - FiFO list (First In First Out)

Queues



Queue Operations

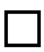
- *Declare*: $\rightarrow$ Q :

  The function value of *Declare(Q)* is an empty queue

Queue Operations

- *Empty*: $\rightarrow$ Q :

  The function *Empty* causes the queue to be emptied and it returns position *End(Q)*



Queue Operations

- *IsEmpty*: Q $\rightarrow$ B :

  The function value *IsEmpty*(*Q*) is *true* if *Q* is empty; otherwise it is *false*

Queue Operations

- *Head*: Q $\rightarrow$ E :

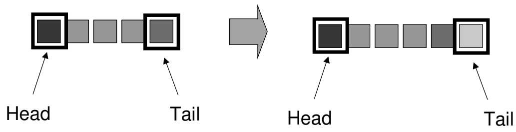
  The function value *Head*(*Q*) is the first element in the list;

  if the queue is empty, the value is undefined

Queue Operations

- *Enqueue*: E $\times$ Q $\rightarrow$ Q :

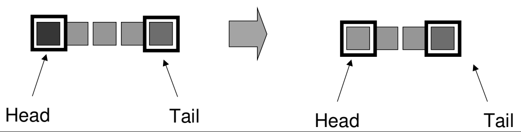
  Enqueue(e, Q)
  Add an element *e* the the tail of the queue



Queue Operations

- *Dequeue*: Q $\rightarrow$ E :

  Dequeue(Q)
  Remove the element from the head of the queue: i.e. return the first element and delete it from the queue



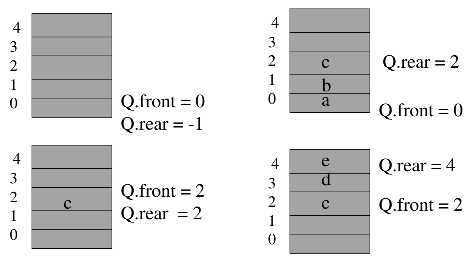
Queue Operations

- *All these operations can be directly implemented using the LIST ADT operations on a queue Q*
- *Again, it may be more efficient to use a dedicated implementation*
- *And, again, it depends what you want: code efficiency or software re-use (i.e. utilization efficiency)*

Implementations

- Array - need a marker for front and rear
  - Empty -> front = 0, rear = -1
  - 3 items -> front = 0, rear = 2
  - What happens when we remove elements?

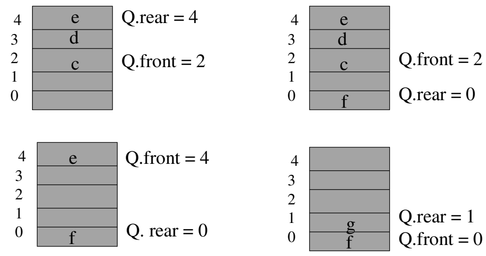
Array Implementation



Circular Array Implementation

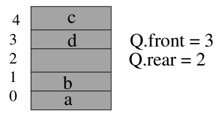
- Circular Array
  - Is this better?
  - Initialisations to Size-1?
  - How do I define Empty / Full?
  - Overflow / Underflow?

Circular Implementation



Overflow & Underflow

- How do I define Empty / Full?
  - Both are defined by q.front = q.rear
- How could we solve this?
  - Sacrifice one element in the array by initialising size to size -1
  - then increment before adding an element,
    if q.rear = q.front -> can't add the element
  - test for remove happened before front is updated.



- if (q.rear ++)= = q.front
  { " Insertion would cause Overflow" }
- if q.rear = = q.front
  { " Removal would cause Underflow" }

Priority Queue

- Intrinsic ordering of the elements does determine the results of its basic operations.
- A priority Queue may be
  - Ascending - items are entered arbitrarily & only the smallest item may be removed.
  - Descending - items are entered arbitrarily & only the largest item may be removed.
- How do we implement insert and delete
- Remember that members of a queue need not be numbers or characters which can be compared directly.

Array implementation of PQ

- Insertion
  - note order of insertion is not tampered with
- Deletion
  - requires a search for the element of highest priority
  - what happens if we delete an element in the centre of the array

Possible Solutions …

- An empty indicator replaces deleted elements. When the array is full compact the array to the front. Inefficient in search.
- Use the empty indicator and insert into the empty slots. Inefficient in insert.
- On each deletion move elements up the array, decrementing rear.
- Maintain array as an ordered circular array