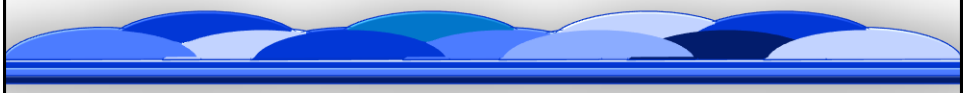
Topic 12 – Abstract Data Types – P2



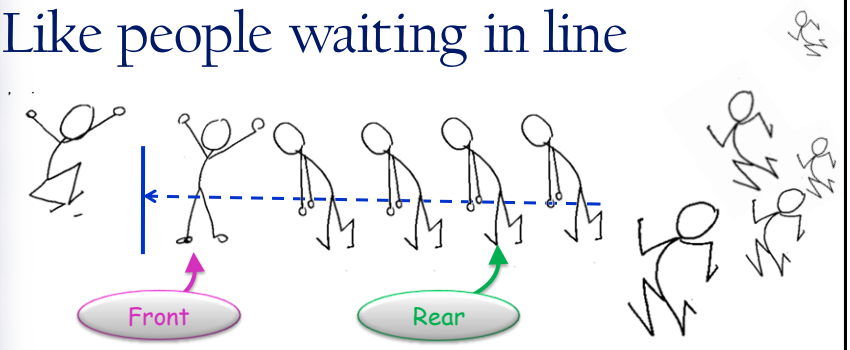
Queues

Queues

Adding to the end and Removing from the front creates a queue

- A queue is an ADT too
  - Elements are added to the end
  - Removed from the front
- Insertions & Deletions follow the FIFO (or LILO) scheme
  - First In First Out
  - Last In Last Out
- Conceptually: A line at a market (who ever enters first, exits first)

Like people waiting in line

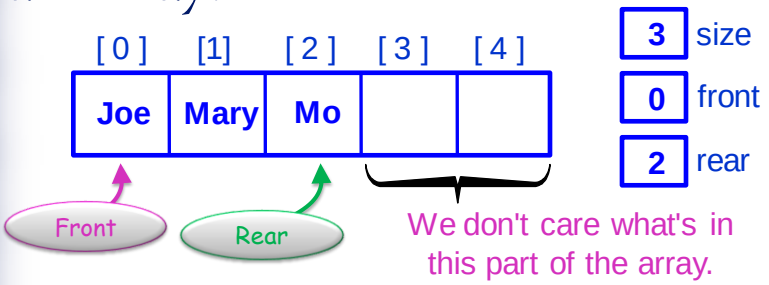


- Objects are added on the rear
- Removed from the front

Queue Operations

Operation	Description
Enqueue (push)	Add an Element to the Rear of the List
Dequeue (pop)	Remove an Element from the front of the list
IsEmpty	Determines whether the queue is empty
Front (Peek)	Returns the element at the front of the queue
Size	Determines the number of elements in the queue
Clear Queue	Deletes the queue

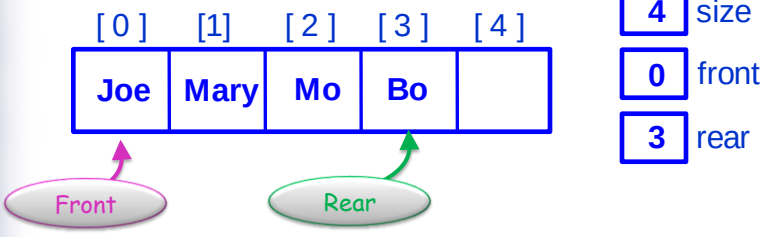
How do we implement this with an Array?



3 size
0 front
2 rear

- We typically want to keep track of the front & rear elements, and the size
- Now, try enqueueing and dequeueing

An Enqueue Operation

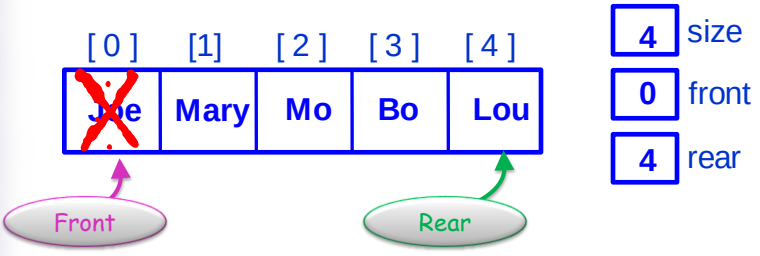


4 size
0 front
3 rear

- Now we want to Enqueue Bo
- We need to move the rear
- We need to change the size and the rear
- Now we want to Enqueue Lou

… our list is full → with a queue we should have an operation to check this

An Dequeue Operation

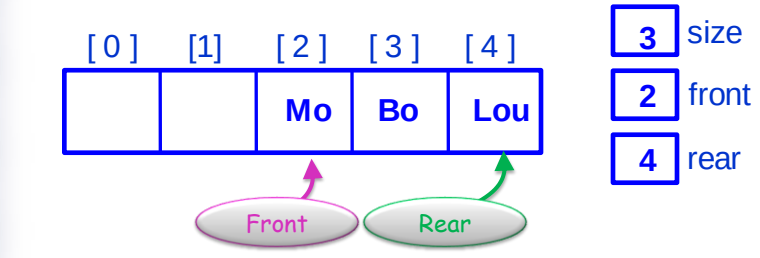


- Now we want to Dequeue
- We Dequeue from the front → so bye Joe
- We need to change the size and the front

- Now we want to Dequeque again

Now we can move Everything forward -- why not?

Now where do we add?



- Try to add Jay
- Wrap around →
- Change size & rear
- Think about how it effects finding size and etc…

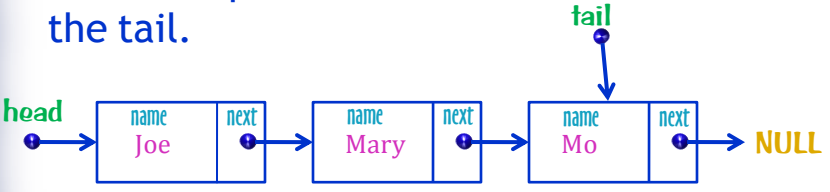
Other Operations

- IsEmpty
  → size = 0
- Front
  → return front
- Size
  → check if rear > front then size = rear – front + 1
  else arSize - (front – rear + 1)
  → count blanks and subtract from ARSIZE
- Clear Queue
  → initialize queue
  → set front & rear = 0
- IsFull
  →size = MAX_AR_SIZE

With an Array we need this → Why?

Implementing a Queue as a linked list.

- Need two pointers to track the head and the tail.



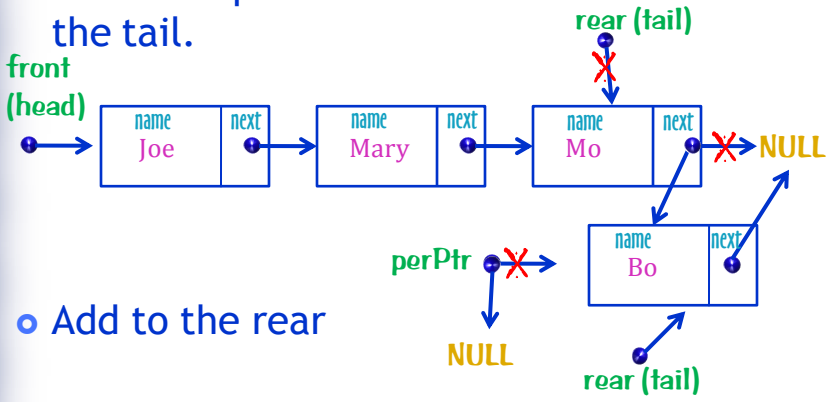
- Which is the front and which is the rear?
- Why?
  → it is harder to remove items from the tail

Queue Operations using a Linked List

- Enqueue
  - → add an element to the REAR of the list
- Dequeue
  - → remove an element from the front of the list
- IsEmpty
  - → check if head points to NULL
- Front(Peek)
  - → Examine the element at the front of the queue
- Size
  - → either count each element
  - → OR keep track as you enqueue and dequeue
- ClearQueue
  - → Delete every element in the list

Enqueue → Linked List

- Need two pointers to track the head and the tail.
- Add to the rear

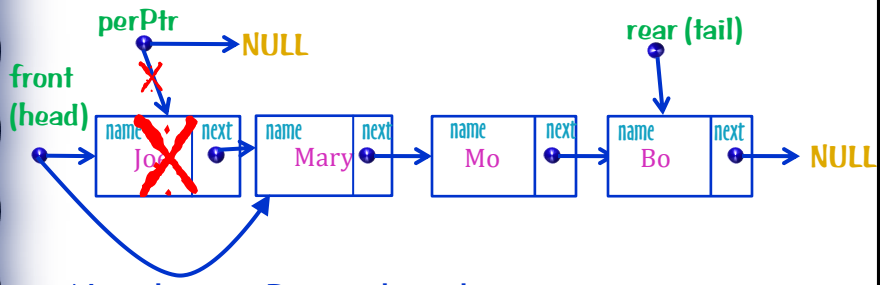


Queue Operations using a Linked List

- Enqueue
  - → add an element to the REAR of the list
- Dequeue
  - → remove an element from the FRONT of the list
- IsEmpty
  - → check if head points to NULL
- Front(Peek)
  - → Examine the element at the front of the queue
- Size
  - → either count each element
  - → OR keep track as you enqueue and dequeue
- ClearQueue
  - → Delete every element in the list

Dequeue → Linked List

- Remove from the front



- Need a perPtr to head
- head = perPt -> next
- Don't forget to delete perPtr
- Reassign perPtr to NULL

Implementing Queue Operations

- IsEmpty
  → check if head == NULL
- Front
  → return the node that head is pointing too
- Size
  → Same as with a stack → loop & count
- Clear Queue
  → Dequeue until head == NULL
- IsFull