Q.A. engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 999999999 beers. Orders a lizard. Orders -1 beers. Orders a sfdeljknesv.

Abstract Data Types

EECS 214

November 2, 2015

So you want a FIFO queue, do you?

What's a FIFO?

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $empty() : \mathsf{FifoQ}$
- $empty?(\mathsf{FifoQ}) : \mathbb{B}$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $\mathit{empty}() : \mathsf{FifoQ}$
- $\mathit{empty?}(\mathsf{FifoQ}) : \mathbb{B}$
- $\mathit{enqueue}(\mathsf{Element}, \mathsf{FifoQ})$
- $\mathit{dequeue}(\mathsf{FifoQ}) : \mathsf{Element}$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $\mathit{empty}() : \mathsf{FifoQ}$
- $\mathit{empty?}(\mathsf{FifoQ}) : \mathbb{B}$
- $\mathit{enqueue}(\mathsf{Element}, \mathsf{FifoQ})$
- $\mathit{dequeue}(\mathsf{FifoQ}) : \mathsf{Element}$

And it has some behavior, *e.g.*:

- $\mathit{empty?}(\mathit{empty}()) = \top$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $empty()$: FifoQ
- $empty?(\text{FifoQ})$: $\mathbb{B}$
- $enqueue(\text{Element}, \text{FifoQ})$
- $dequeue(\text{FifoQ})$: Element

And it has some behavior, *e.g.*:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $empty() : \mathsf{FifoQ}$
- $empty?(\mathsf{FifoQ}) : \mathbb{B}$
- $enqueue(\mathsf{Element}, \mathsf{FifoQ})$
- $dequeue(\mathsf{FifoQ}) : \mathsf{Element}$

And it has some behavior, *e.g.*:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$
- $q \leftarrow empty()$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $empty() : \mathsf{FifoQ}$
- $empty?(\mathsf{FifoQ}) : \mathbb{B}$
- $enqueue(\mathsf{Element}, \mathsf{FifoQ})$
- $dequeue(\mathsf{FifoQ}) : \mathsf{Element}$

And it has some behavior, *e.g.*:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$
- $q \leftarrow empty()$
  $enqueue(a, q); enqueue(b, q)$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $empty() : \mathsf{FifoQ}$
- $empty?(\mathsf{FifoQ}) : \mathbb{B}$
- $enqueue(\mathsf{Element}, \mathsf{FifoQ})$
- $dequeue(\mathsf{FifoQ}) : \mathsf{Element}$

And it has some behavior, *e.g.*:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$
- $q \leftarrow empty()$
  $enqueue(a, q); enqueue(b, q)$
  $a' \leftarrow dequeue(q); b' \leftarrow dequeue(q)$

So you want a FIFO queue, do you?

What's a FIFO? Well, it has some operations:

- $empty()$: FifoQ
- $empty?(\text{FifoQ})$: $\mathbb{B}$
- $enqueue(\text{Element}, \text{FifoQ})$
- $dequeue(\text{FifoQ})$: Element

And it has some behavior, *e.g.*:

- $empty?(empty()) = \top$
- $enqueue(e, q); empty?(q) = \bot$
- $q \leftarrow empty()$
  $enqueue(a, q); enqueue(b, q)$
  $a' \leftarrow dequeue(q); b' \leftarrow dequeue(q)$
  $a' = a \wedge b' = b \wedge empty?(q) = \top$

But what *is* it?

It doesn't matter.

HOW CAN IT NOT MATTER?

Let's use one and see

– adt.rkt –

ADTs can have multiple implementations

Like you saw on the exam! Two possible FIFO implementations:

- linked list
- ring buffer

Linked list FIFO

```
(define-struct list-fifo-cell [first rest])
(define-struct list-fifo      [front back])

; A ListFifoList is one of:
; - '()
; - (make-list-fifo-cell Element ListFifoList)

; A ListFifo is
;   (make-list-fifo
;     ListFifoList
;     (make-list-fifo-cell Element '()))
; where either
; - both fields are '(), or
; - the `back' is the last cell of `front'
```

Ring buffer FIFO

```
(define-struct ring-fifo [front back elements])

; A RingFifo is
;   (make-ring-fifo N N [Vector-of Element])
;
; where `front' and `back' are valid indices
; for `elements', and one of:
;  - front = back means it's empty
;  - front < back means the FIFO comprises
;    elements [front, back)
;  - front > back means the FIFO comprises
;    elements [front, size) then [0, back)
```