COMP 250

Lecture 8

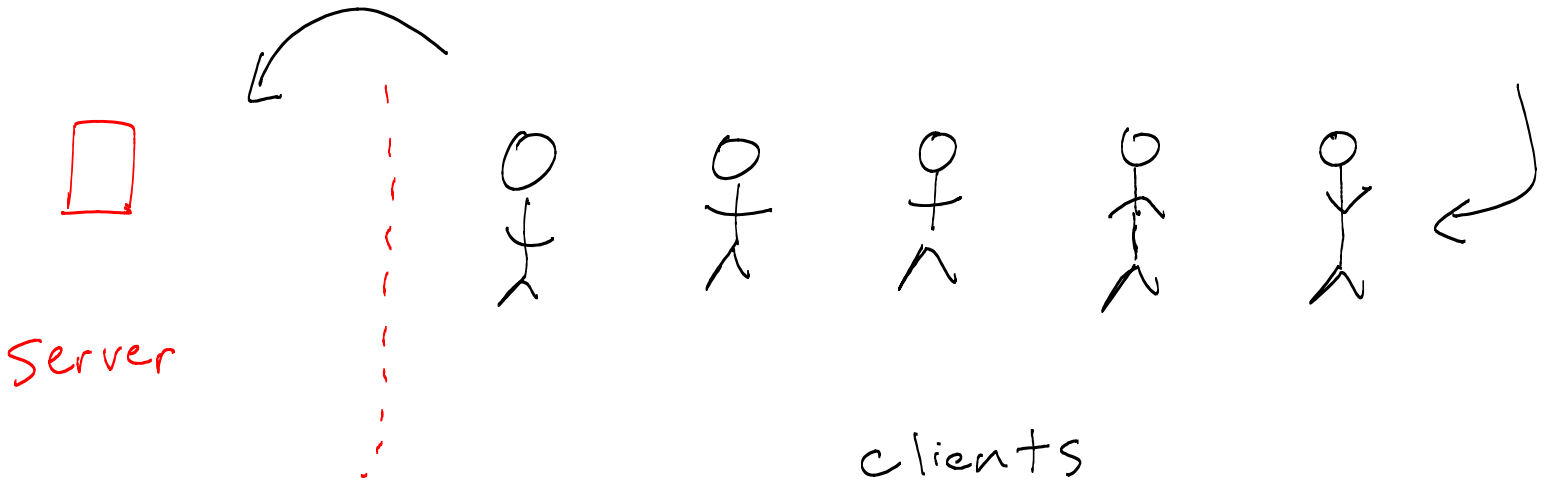
queue ADT

Sept. 23, 2016

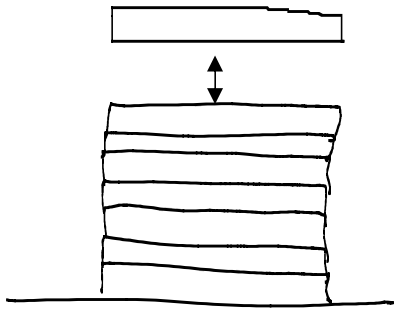
Queue

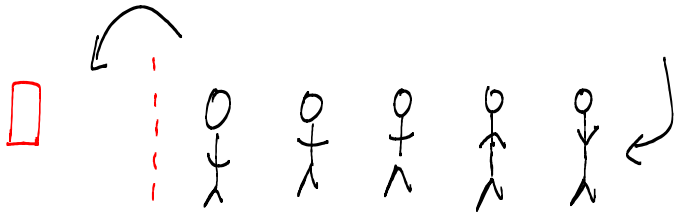
dequeue
(remove from
front)

enqueue
(add at back)



Queues are heavily used in OS (operating systems) e.g. process scheduling.





Stack

push(e)

pop()

LIFO
(last in,
first out)

Queue

enqueue(e)

dequeue()

FIFO
(first in,
first out)

ADT's (abstract data types)

- List

  add(i,e), remove(i), get(i), set(i), …..

- Stack

  push, pop(), ..

- Queue

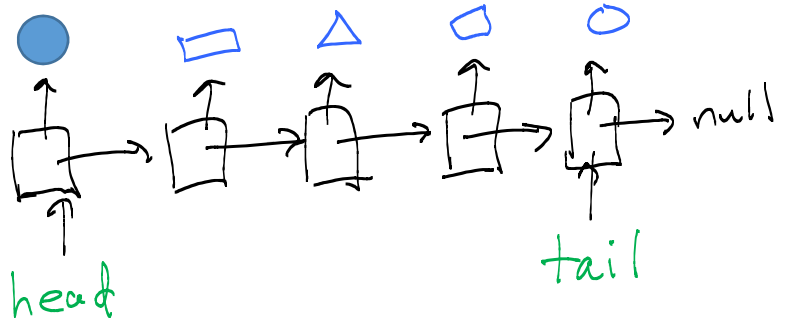
  enqueue(e), dequeue()

Although stacks and queues consist of a finite ordered set of elements, strictly speaking, they are not lists since their operations do not allow one to index directly to the arbitrary elements.

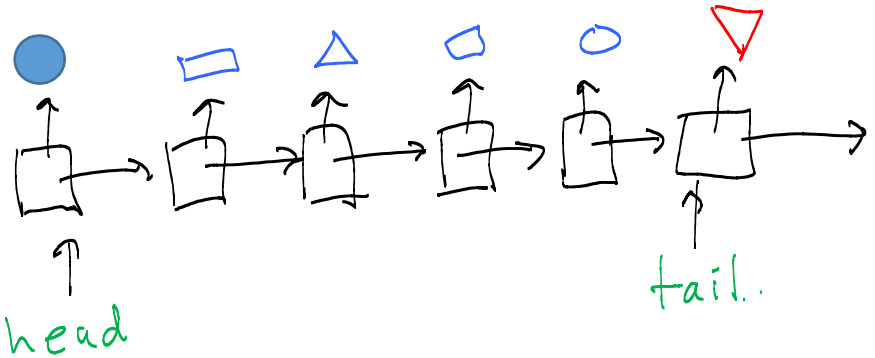
Queue Example

enqueue(a)	a
enqueue(b)	ab
dequeue( )	b
enqueue(c)	bc
enqueue(d)	bcd
enqueue(e)	bcde
dequeue( )	cde
enqueue(f)	cdef
dequeue( )	def
enqueue(g)	defg

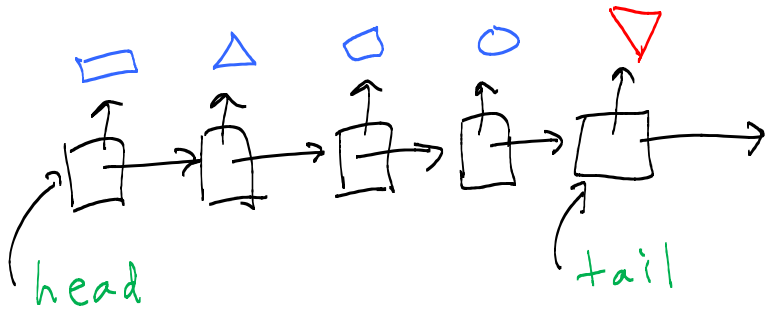
Implementing a queue with a singly linked list.



enqueue(▽) = addLast(▽)



dequeue( ) = removeFirst( )



Implementing a queue with an array list. (1st attempt)

length = 4

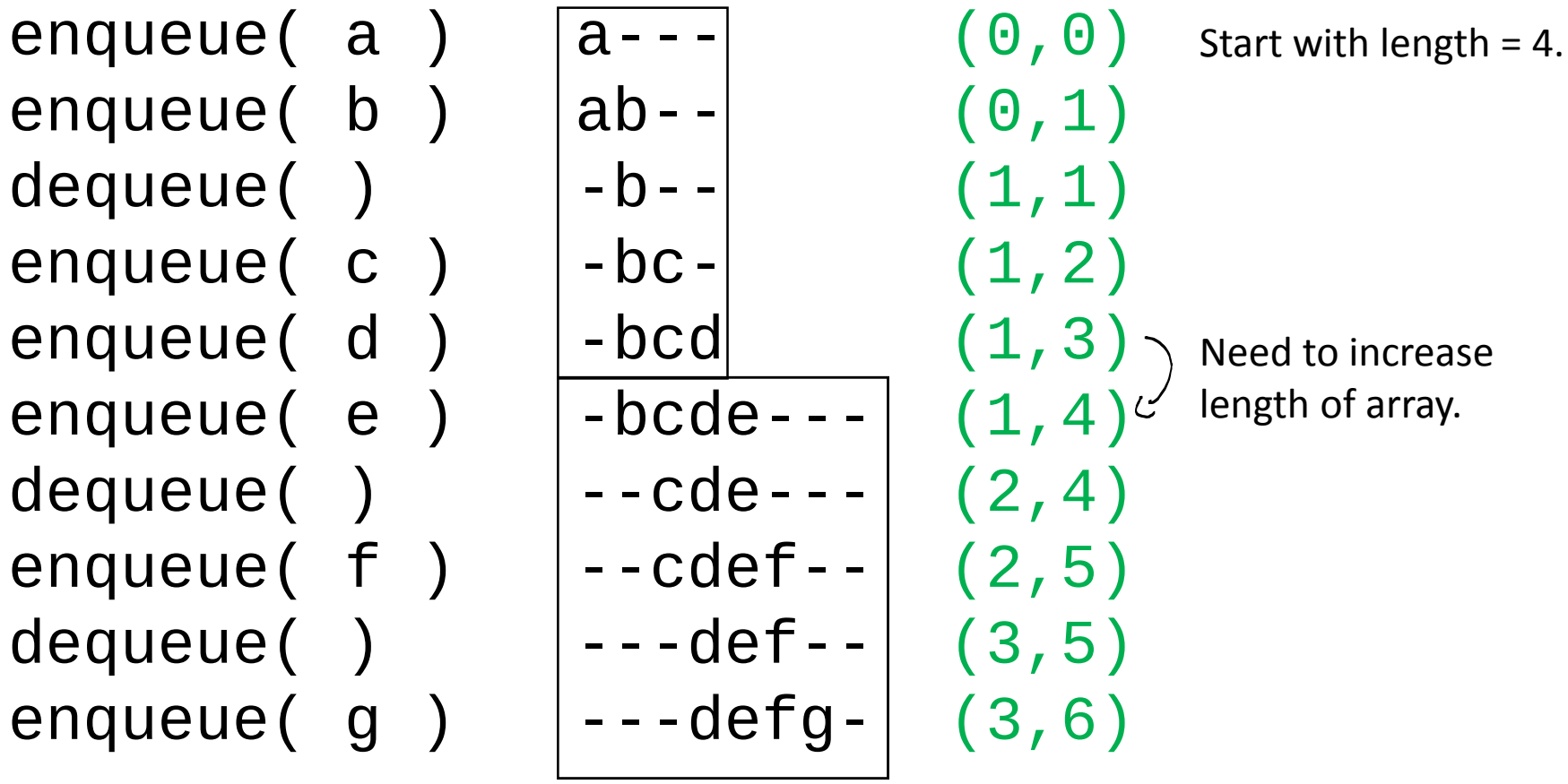
0123

indices

enqueue(a)	a---	
enqueue(b)	ab--	
dequeue( )	b---	removeFirst (& shift)
enqueue(c)	bc--	
enqueue(d)	bcd-	
enqueue(e)	bcde	
dequeue( )	cde-	removeFirst (& shift)
enqueue(f)	cdef	
dequeue( )	def-	removeFirst (& shift)
enqueue(g)	defg	

Implementing a queue with an array. (2nd attempt)

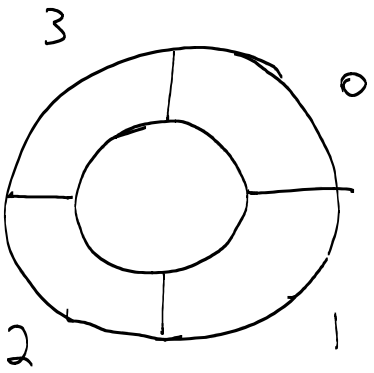
Use head and tail indices
(tail = head + size – 1)



Circular array

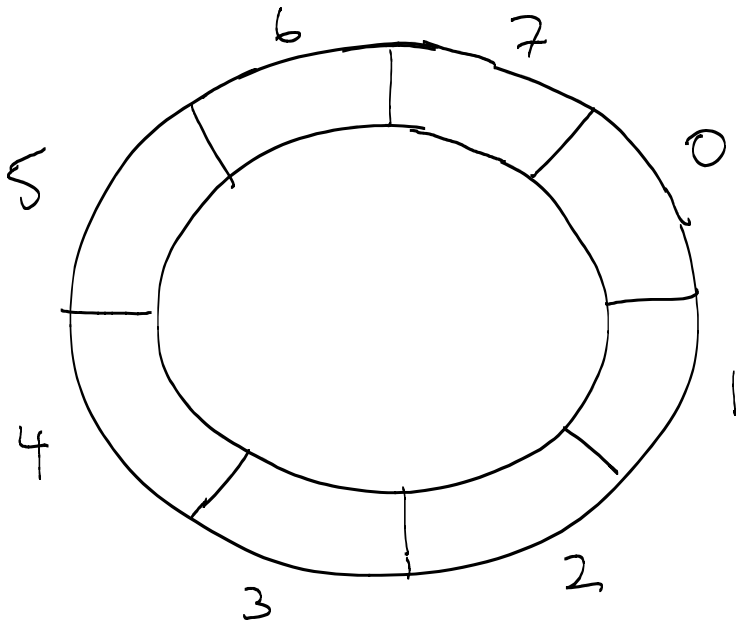
length = 4

```
0123
```



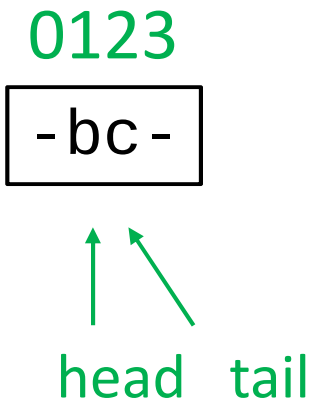
length = 8

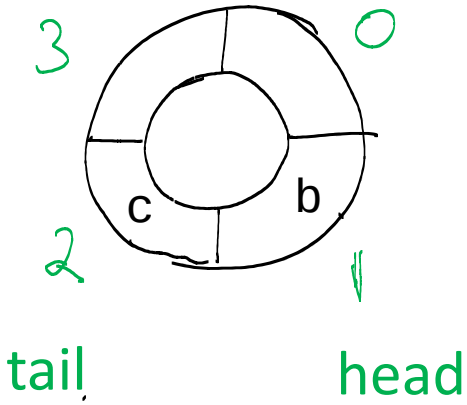
```
01234567
```

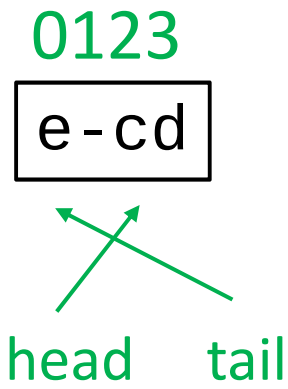


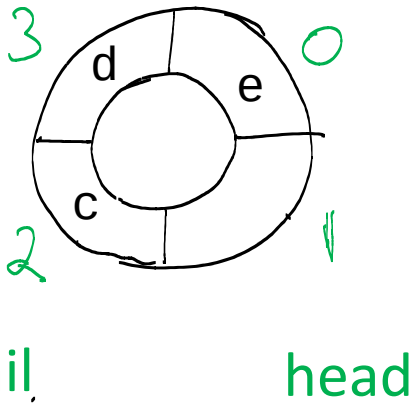
Circular array

tail = (head + size – 1) % length









Implementing a queue with a circular array (GOOD)

tail = (head + size – 1) % length

	queue	array	(head, tail, size)
enqueue(a)	a	a---	(0,0, 1)
enqueue(b)	ab	ab--	(0,1, 2)
dequeue()	b	-b--	(1,1, 1)
enqueue(c)	bc	-bc-	(1,2, 2)
enqueue(d)	bcd	-bcd	(1,3, 3)
enqueue(e)	bcde	**ebcd**	(1,0, 4)

The code below does not properly handle the case that size == 1. See lecture notes where this has been corrected. Note that, when size == 0, head is different from tail. Also, when queue is initialized, head == 0 and tail == length – 1.

```
dequeue( ){
   //  check that size >=1  (omitted)
   element = queue[ head ]
   if (size > 1)
       head = (head + 1) % length
   size = size – 1
   // don't adjust tail
   return  element
}
```

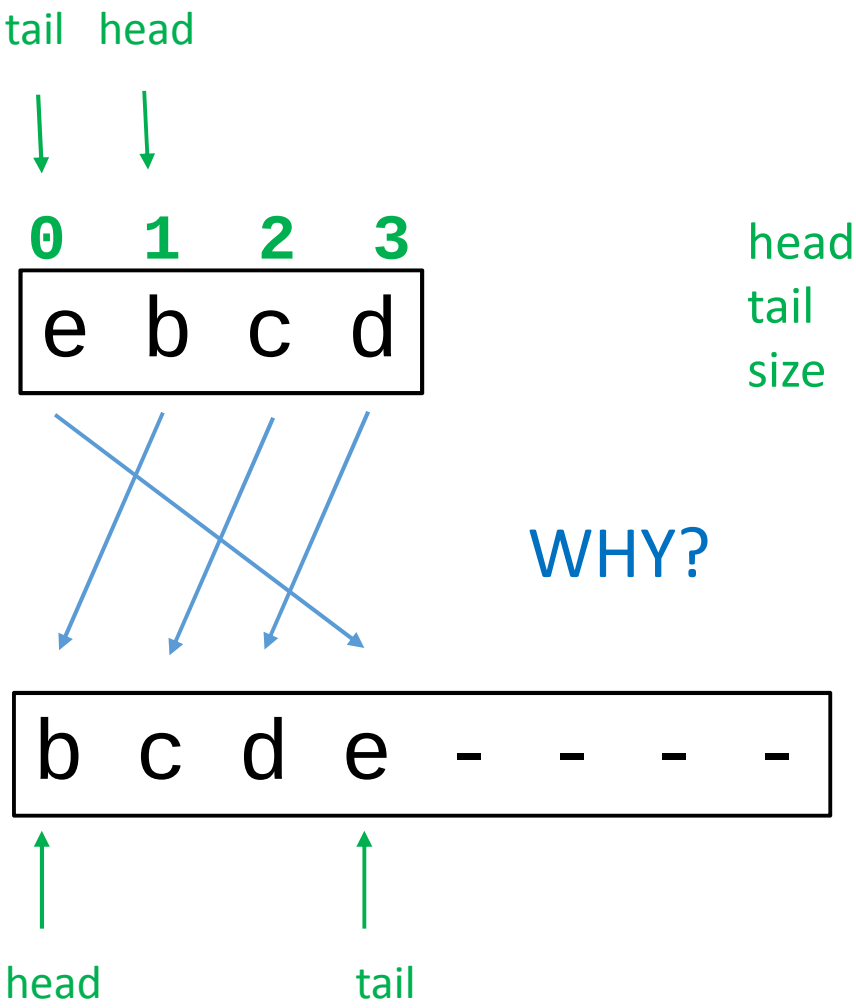
How to enqueue if the array is full ?

```
enqueue( element ){
    if ( size == length)
        increase length of array and rearrange
    size = size + 1
    tail = (tail + 1) % length
    queue[tail] = element
}
```

The example shown in the following slide is slightly different from the one used in the lecture.

Please see lecture notes for further discussion of enqueueing an element when the array is full.

increase length of array and rearrange



```
enqueue( element ){
    if ( size == length)  {
        //  increase length of array

        create a bigger array called tmp
        for i = 0  to queue.length  - 1
            tmp[i] = queue[ (head + i) %  queue.length ]
        head = 0
        tail    = size-1
    }
    size = size + 1
    tail = (tail + 1) % length
    queue[tail] = element
}
```

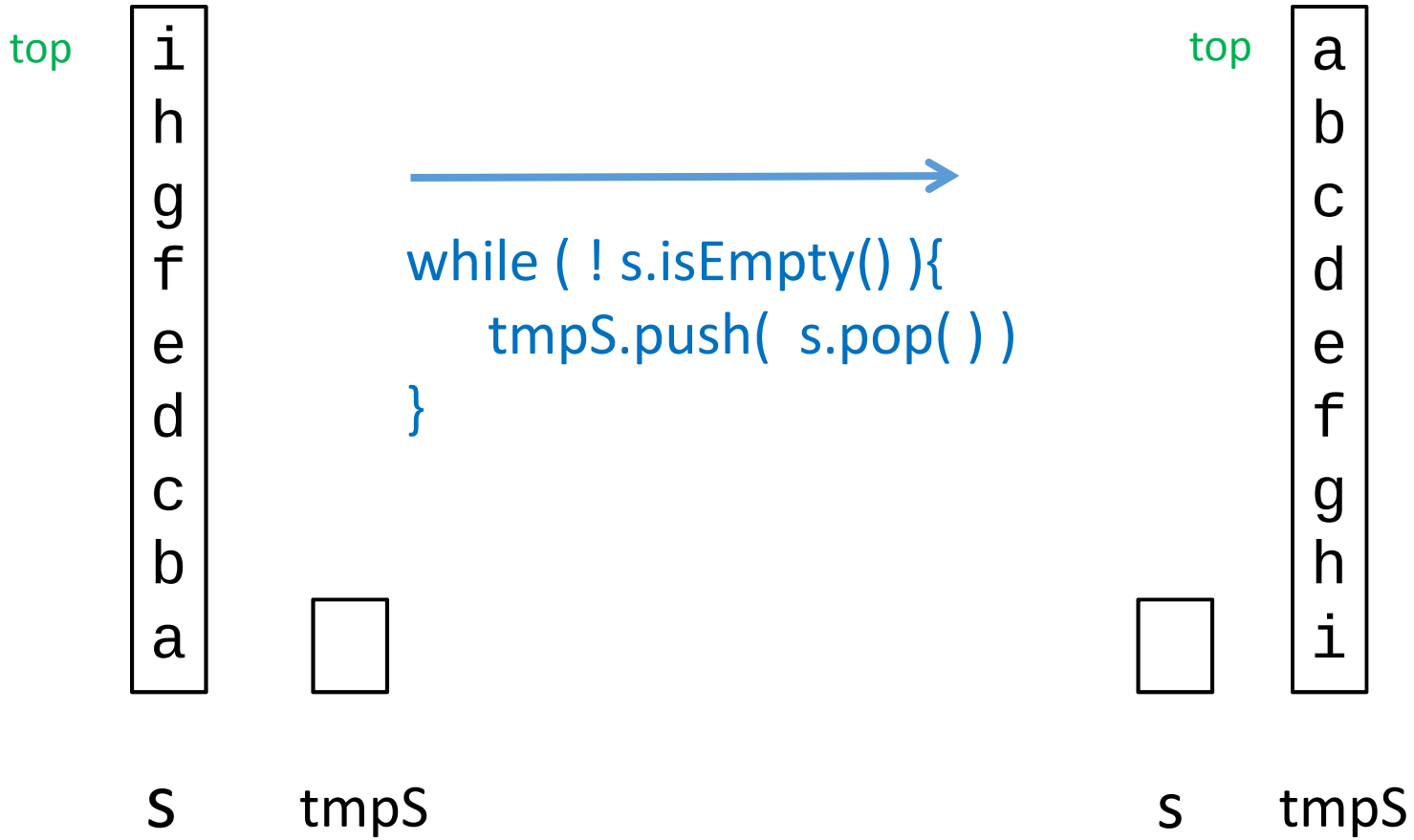
Exercise: Use stack(s) to implement a queue.

```
 enqueue( e ){      // add element
            :
}

 dequeue( ) {     // remove 'oldest' element
            :
}
```

Write pseudocode for these two methods that uses a stack, namely use the operations push(e) , pop(), isEmpty() .

Hint for Exercise



Some possibly confusing terminology (ADT, Java API, Java `interface`)

- List `interface`

  add(i,e), remove(i), get(i), set(i), …..

- Stack `class`

  push, pop(), ..

- Queue `interface`

  offer(e), poll (), ….