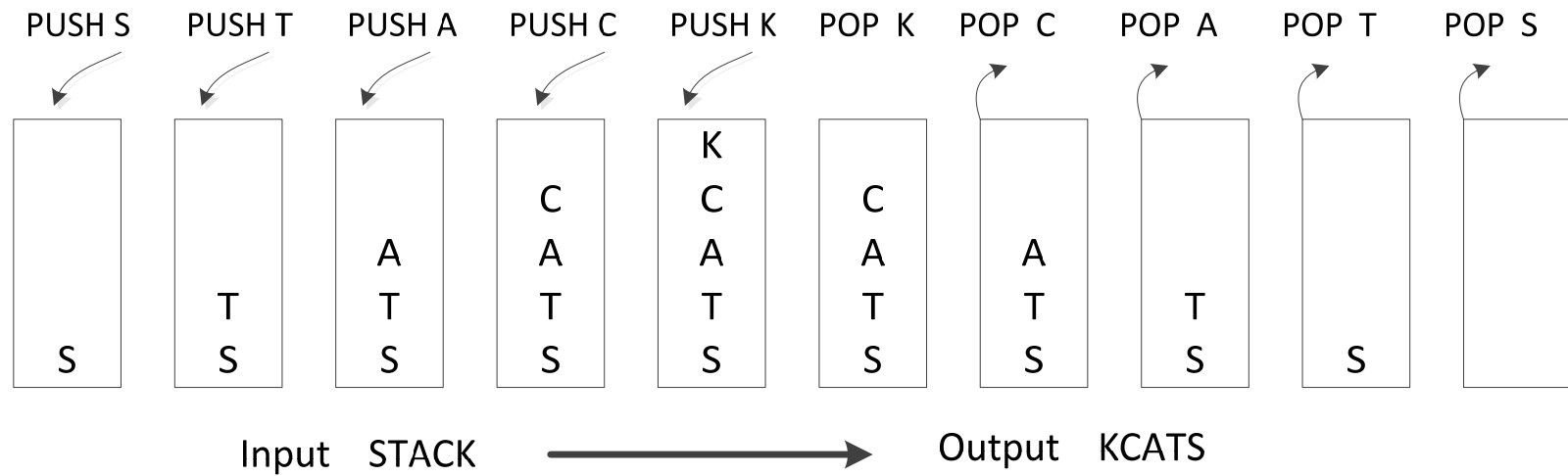


Data Structures and Algorithms

Chapter 6

Stacks

- Illustration



Stacks

- Stack ADT supports the following operations:
 - `push(e)`: Adds element `e` to the stack top.
 - `pop()`: Removes and returns the top element from the stack. Returns null if the stack is empty.
 - `top()`: Returns the top element of the stack without removing it. Returns null if the stack is empty.
 - `size()`: Returns the number of elements in the stack.
 - `isEmpty()`: Returns true if the stack is empty and false otherwise.

Stacks

- Illustration

Operation	Return Value	Stack Contents → top
push(10)	-	(10)
push(20)	-	(10, 20)
push(5)	-	(10, 20, 5)
size()	3	(10, 20, 5)
top()	5	(10, 20, 5)
pop()	5	(10, 20)
push(30)	-	(10, 20, 30)
pop()	30	(10, 20)
pop()	20	(10)
pop()	10	()
isEmpty()	true	()
pop()	null	()

Stacks

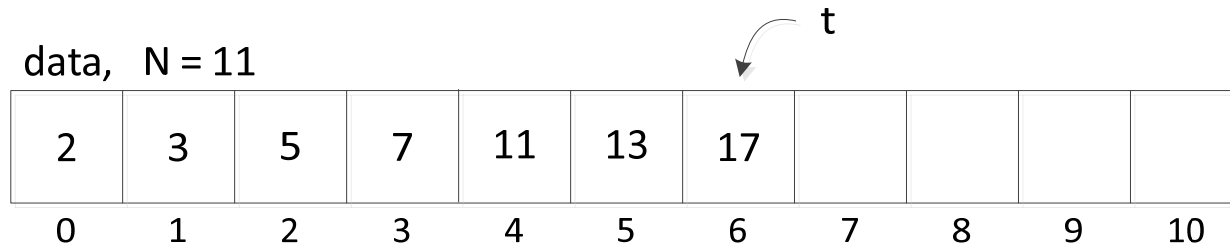
- Java's stack (*java.util.Stack* class)

Stack ADT in Textbook	Class java.util.Stack
size()	size()
isEmpty()	empty()
push(e)	push(e)
pop()	pop()
top()	peek()

- JavaStackDemo.java

Stacks

- Array-based implementation
 - The bottom element is stored in *data*[0].
 - The top element is stored in *data*[*t*], $0 \leq t < N$.
 - When the stack is empty, by convention, $t = -1$.



- ArrayStack.java

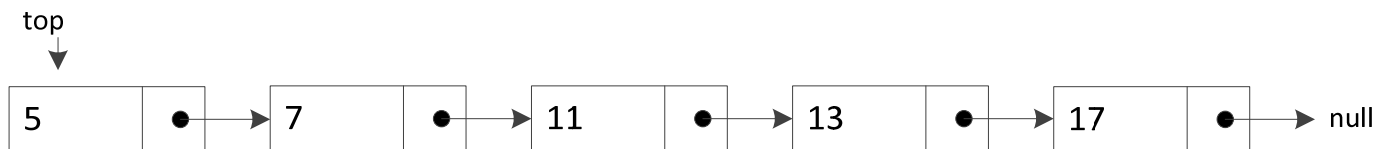
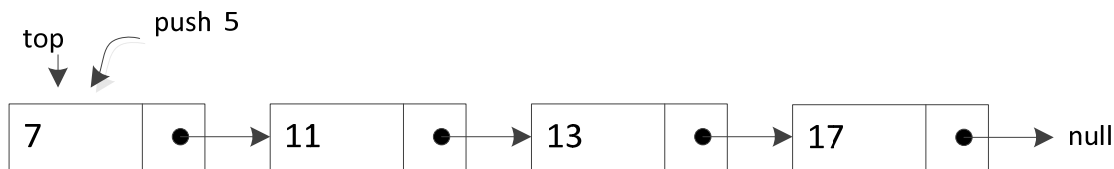
Stacks

- Running times of array-based implementation

Method	Running Time
size()	$O(1)$
isEmpty()	$O(1)$
push(e)	$O(1)$
pop()	$O(1)$
top()	$O(1)$

Stacks

- Stack implementation using singly-linked list.
 - Stack top element is stored at the head of a list.
 - All operations take $O(1)$.



- LinkedStack.java* code

Stacks

- Reversing array elements

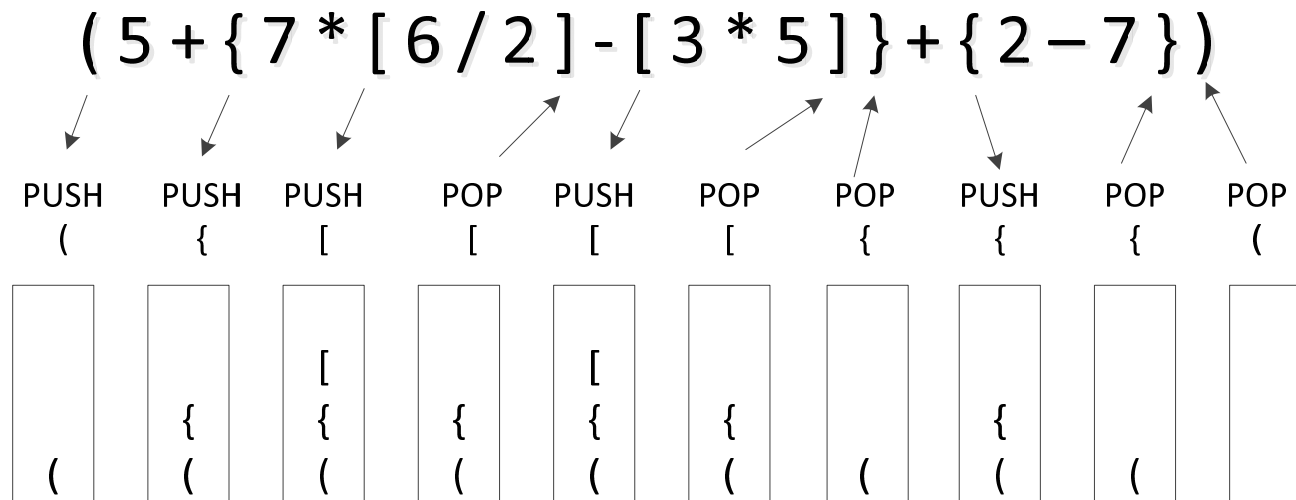
```
1  public static <E> void reverse(E[ ] a) {  
2      Stack<E> buffer = new ArrayStack<>(a.length);  
3      for (int i=0; i < a.length; i++)  
4          buffer.push(a[i]);  
5      for (int i=0; i < a.length; i++)  
6          a[i] = buffer.pop();  
7  }
```

Stacks

- Matching parentheses
 - Scan the expression one character at a time from left to right
 - If the character is an opening delimiter, push it to the stack
 - If the character is a closing delimiter:
 - Pop a delimiter from the stack
 - Compare that with the closing delimiter being scanned
 - If they are a matching pair (for example, a left square bracket and a right square bracket), continue
 - Else, the expression is invalid

Stacks

- Matching parentheses (continued)

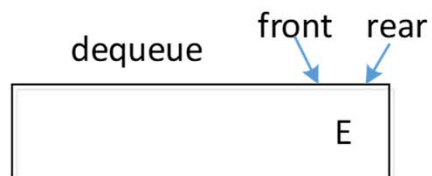
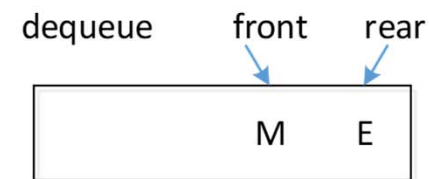
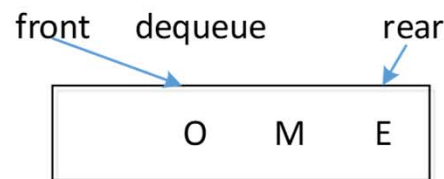
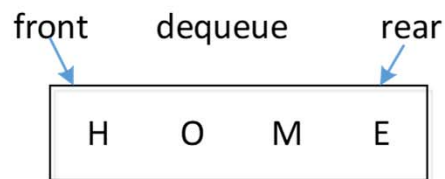
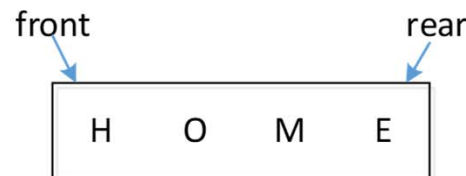
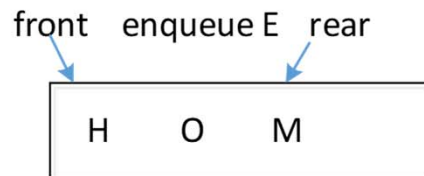
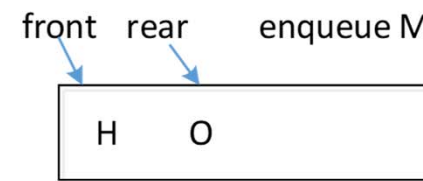
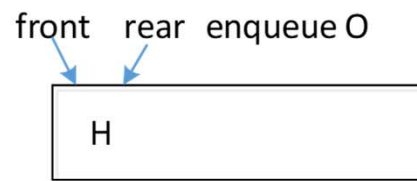
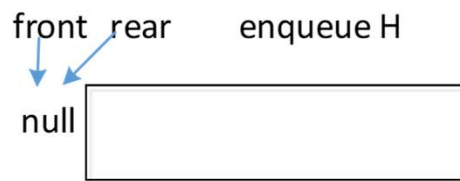


Queues

- A queue has a linear data structure (like stack)
- An object is added to one end called *rear*, and removed from the other end called *front*.
- FIFO (first in first out).
- Adding is called *enqueue*
- Removing is called *dequeue*

Queues

- Illustration



Queues

- Queue ADT operations:
 - enqueue(*e*): Adds element *e* to the back of queue.
 - dequeue(): Remove and returns the first element from the queue. Returns null if the queue is empty.
 - first(): Returns the first element of the queue, without removing it. Returns null if the queue is empty.
 - size(): Returns the number of elements in the queue.
 - isEmpty(): Returns true if the queue is empty and false otherwise.

Queues

- Illustration of operations:

Operation	Return Value	first \leftarrow Q \leftarrow last
enqueue(10)	-	(10)
enqueue(20)	-	(10, 20)
enqueue(5)	-	(10, 20, 5)
size()	3	(10, 20, 5)
dequeue()	10	(20, 5)
enqueue(30)	-	(20, 5, 30)
dequeue ()	20	(5, 30)
dequeue ()	5	(30)
dequeue ()	30	()
isEmpty()	true	()
dequeue()	null	()

Queues

- Java has the *java.util.Queue* interface.
- Java Queue interface operations:

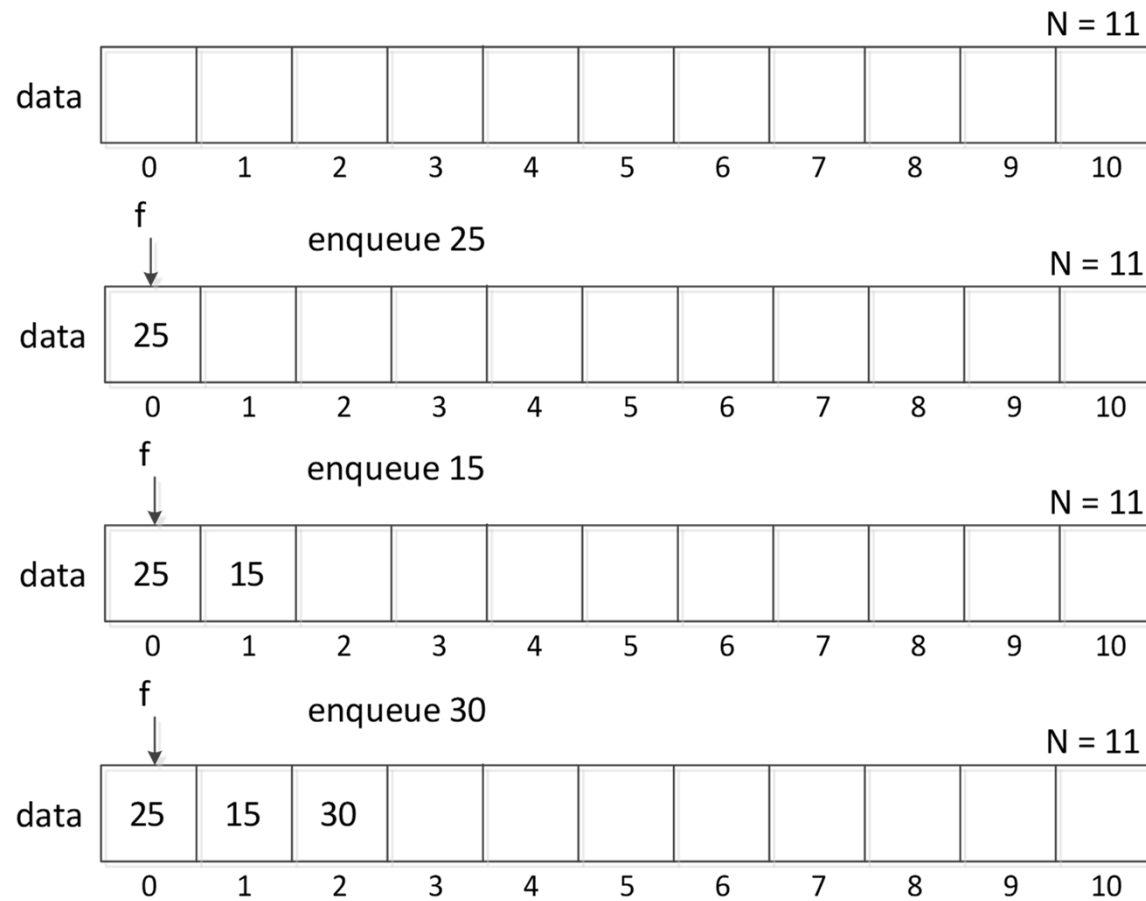
Queue ADT	Interface java.util.Queue	
	throws exception	returns special value
enqueue(e)	add(e)	offer(e)
dequeue()	remove()	poll()
first()	element()	peek()
size()	size()	
isEmpty()	isEmpty	

Queues

- In Java:
 - LinkedList class implements List and Deque interfaces.
 - Deque interface extends Queue interface.
- JavaQueueDemo.java

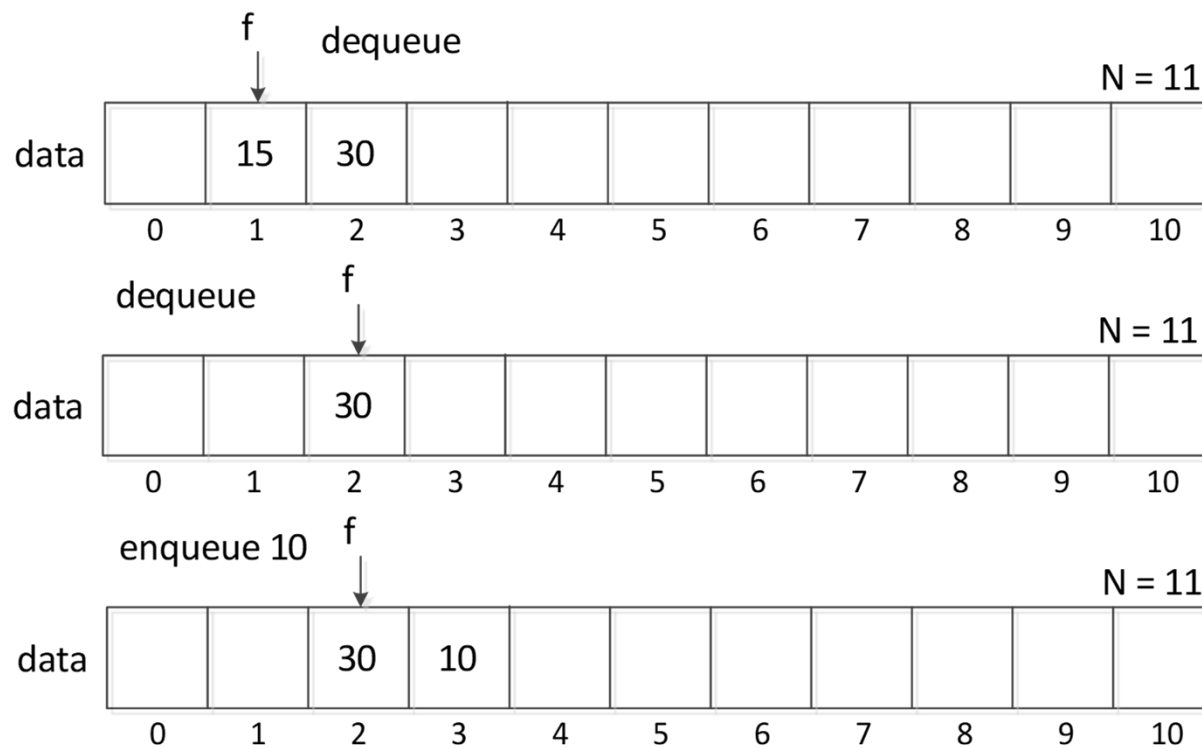
Queues

- Array-based implementation (*ArrayQueue.java*)



Queues

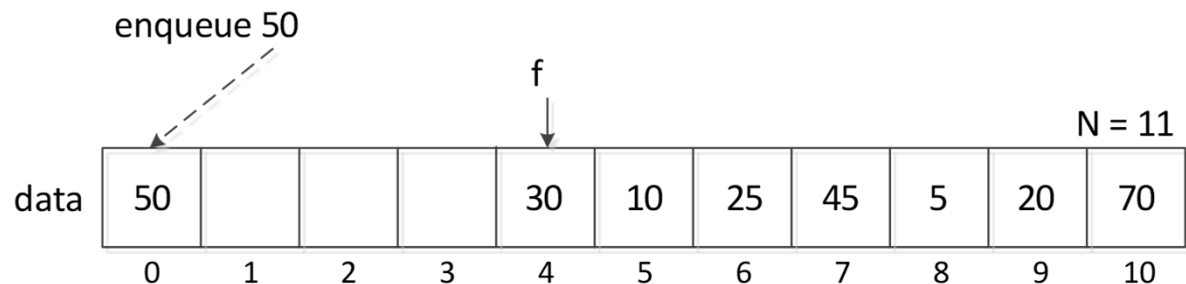
- Array-based implementation (continued)



Queues

- Array-based implementation (continued)
- Elements are added in a wrap around manner:

```
1 public void enqueue(E e) throws IllegalStateException {  
2     if (sz == data.length)  
        throw new IllegalStateException("Queue is full");  
3     int avail = (f + sz) % data.length; // use modular arithmetic  
4     data[avail] = e;  
5     sz++;  
6 }
```



Queues

- Singly linked list-based implementation

```
1 public class LinkedQueue<E> implements Queue<E> {  
2     private SinglyLinkedList<E> list = new SinglyLinkedList<>();  
3     public LinkedQueue() { }  
4     public int size() { return list.size(); }  
5     public boolean isEmpty() { return list.isEmpty(); }  
6     public void enqueue(E element) { list.addLast(element); }  
7     public E first() { return list.first(); }  
8     public E dequeue() { return list.removeFirst(); }  
9 }
```

Queues

- Double-ended queue, called *deque* (pronounced “deck”).
 - Allows insertion and deletion at both ends.
 - Can be used as a stack or as a queue
- Queue and Deque (in Java)

Queue Method	Equivalent Deque Method
add(e)	addLast(e)
offer(e)	offerLast(e)
remove()	removeFirst()
poll()	pollFirst()
element()	getFirst()
peek()	peekFirst()

Queues

- Stack and Deque (in Java)

Stack Method	Equivalent Deque Method
push(e)	addFirst(e)
pop()	removeFirst()
peek()	peekFirst()

- *java.util.Deque* interface.
- *java.util.ArrayDeque* implements *Deque* interface.

References

- M.T. Goodrich, R. Tamassia, and M.H. Goldwasser, “Data Structures and Algorithms in Java,” Sixth Edition, Wiley, 2014.
- Oracle online documentation