

# Lecture 4

## (Singly) Linked Lists

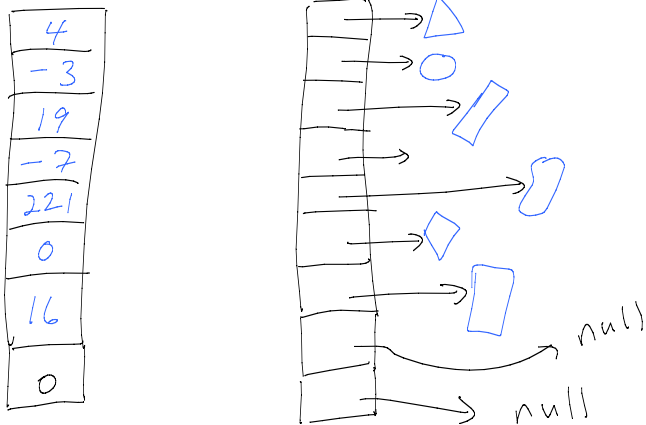
# Eclipse tutorials (beginners only)

Monday Jan. 17 2:30 - 3:30

Wed. Jan 19 1:30 - 2:30

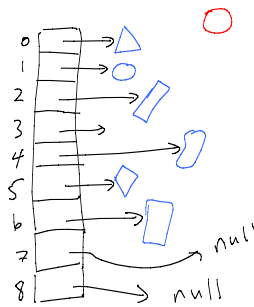
Trottier 3120

## Arrays

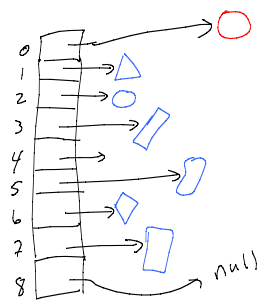


Limitations of Arrays:  
What if you want to add an element at the front?

BEFORE



AFTER

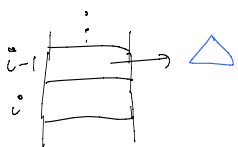


Add/insert **newElement** at front

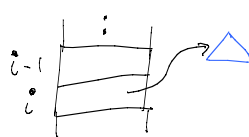
for  $i = N$  down to 1  
 $a[i] \leftarrow a[i-1]$  ]  $N$  steps

$a[0] \leftarrow \text{newElement}$

BEFORE

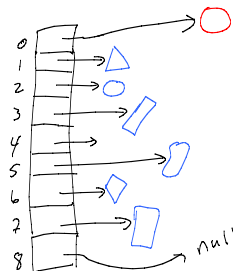


AFTER

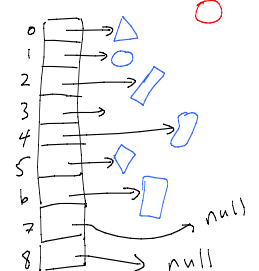


Same problem arises if we remove front element.

BEFORE



AFTER

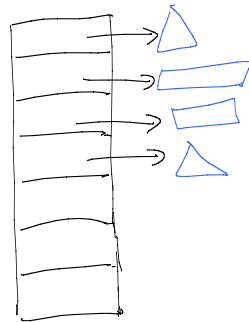


Again,  $N$  Steps required.

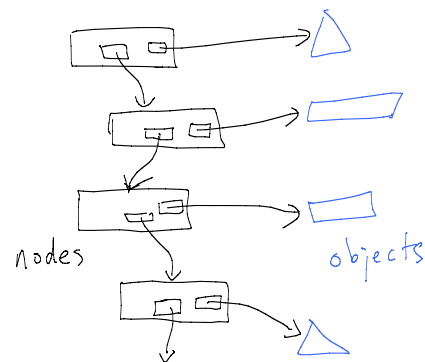
Adding / Removing from the "other end" of the array is fast (constant # steps, is. independent of  $N$ )

No shifts necessary.

## Arrays

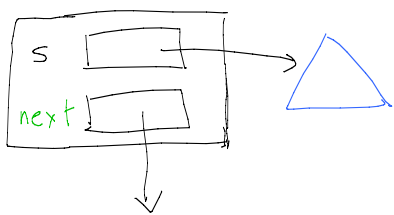


## Linked Lists (chain of "nodes")



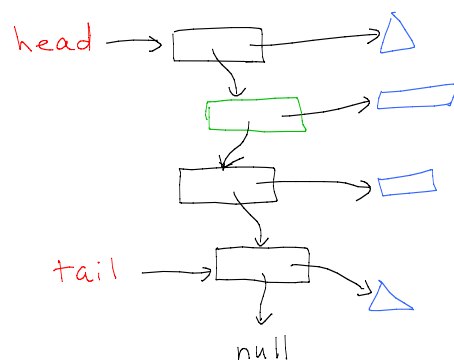
## Singly Linked List Node

Example: a list of Shape objects



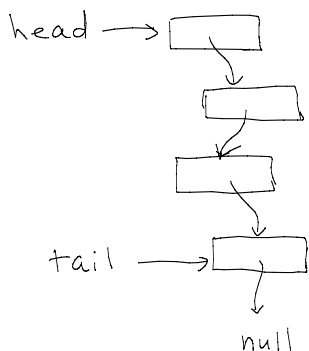
```
class SNode {
    Shape s;
    SNode next;
}
```

## Singly Linked List (of Nodes)



Q: What does `head.next` refer to?

## Linked List



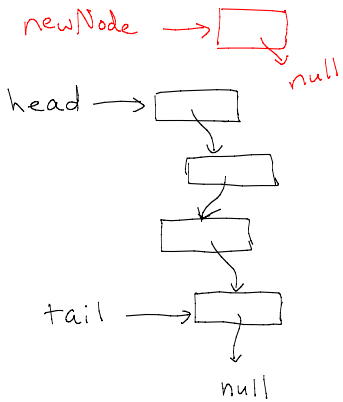
Many linked list algorithms ignore the data that is stored or referenced at each node,

## Algorithms

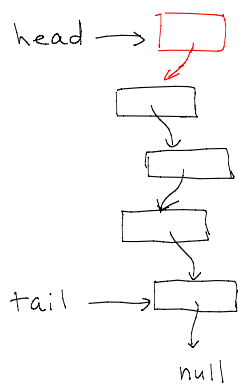
- add First ( new Node )
- remove First ( )
- add Last ( new Node )
- remove Last ( )

Add a new first node

BEFORE



AFTER

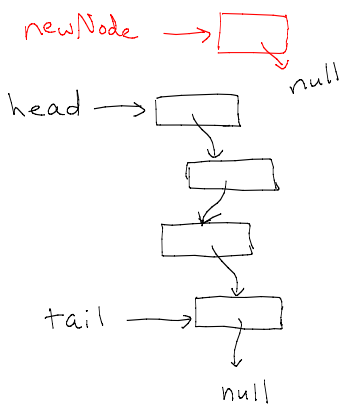


```
addFirst ( newNode ) {
    newNode.next ← head
    head ← newNode
}
```

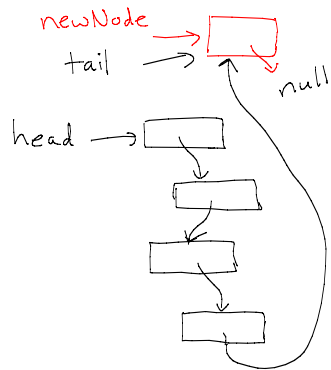
Note that the order matters!

Add a new last node

BEFORE



AFTER

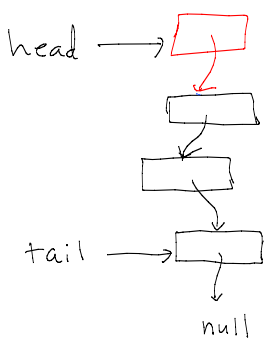


```
addLast ( newNode ) {
    tail.next ← newNode
    tail ← tail.next
}
```

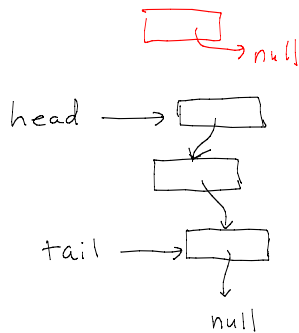
Again, the order matters.

Remove the first node (assuming head ≠ null)

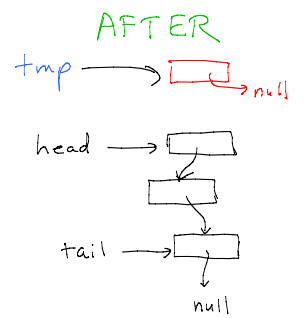
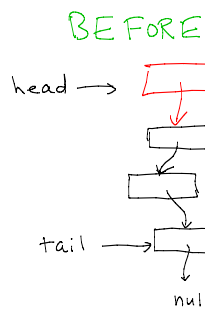
BEFORE



AFTER

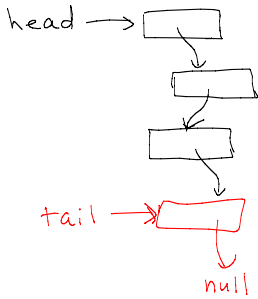


```
removeFirst () {
    tmp ← head
    head ← head.next
    tmp.next ← null
}
```

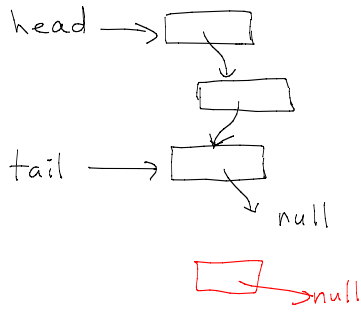


Remove the last node

BEFORE



AFTER



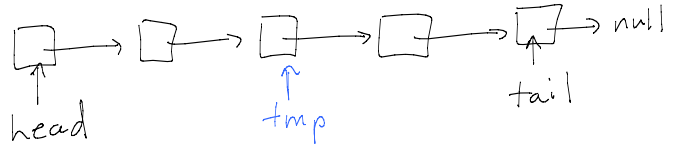
Problem: you need to know the node that points to tail.

Basic idea we need:

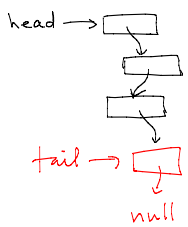
```

visit All () {
    tmp ← head
    while (tmp ≠ null) {
        tmp ← tmp.next
    }
}

```



BEFORE

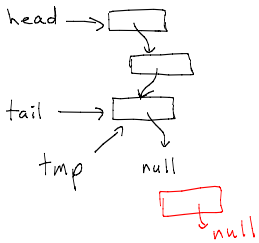


```

Remove Last () {
    tmp ← head
    while (tmp.next ≠ tail) {
        tmp ← tmp.next
    }
    tail ← tmp
    tail.next ← null
}

```

AFTER



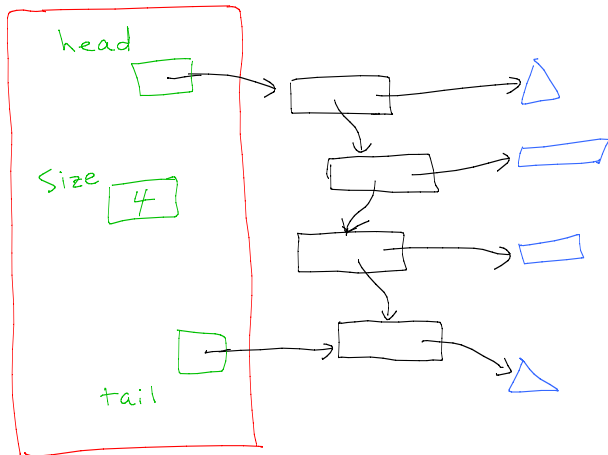
// This code assumes more than 1 node in list.

Algorithms

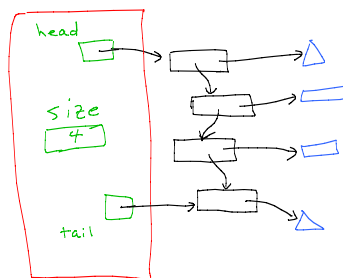
	array	linked list
add First	N	1
remove First	N	1
add Last	1	1
remove Last	1	N

Here we do not distinguish 1 vs 2 vs 3 operations.

Singly Linked List



Singly Linked List

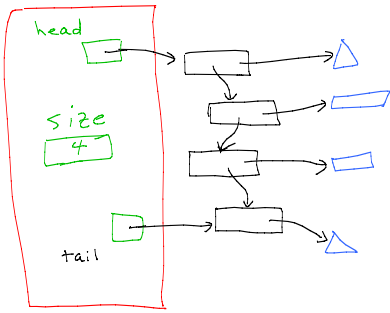


```

class SLinkedList
    SNode head
    SNode tail
    int size
}

```

How many objects ?



$$1 + 4 + 4 = 9$$

