sit with anyone that looks friendly

CMPS 12B/M
Introduction to Data Structures

- Instructor: Nathan Whitehead

Priority Queue

7	
6	
5	
4	
3	22
2	57
1	189
0	212

Remove

7	
6	
5	
4	
3	
2	57
1	189
0	212

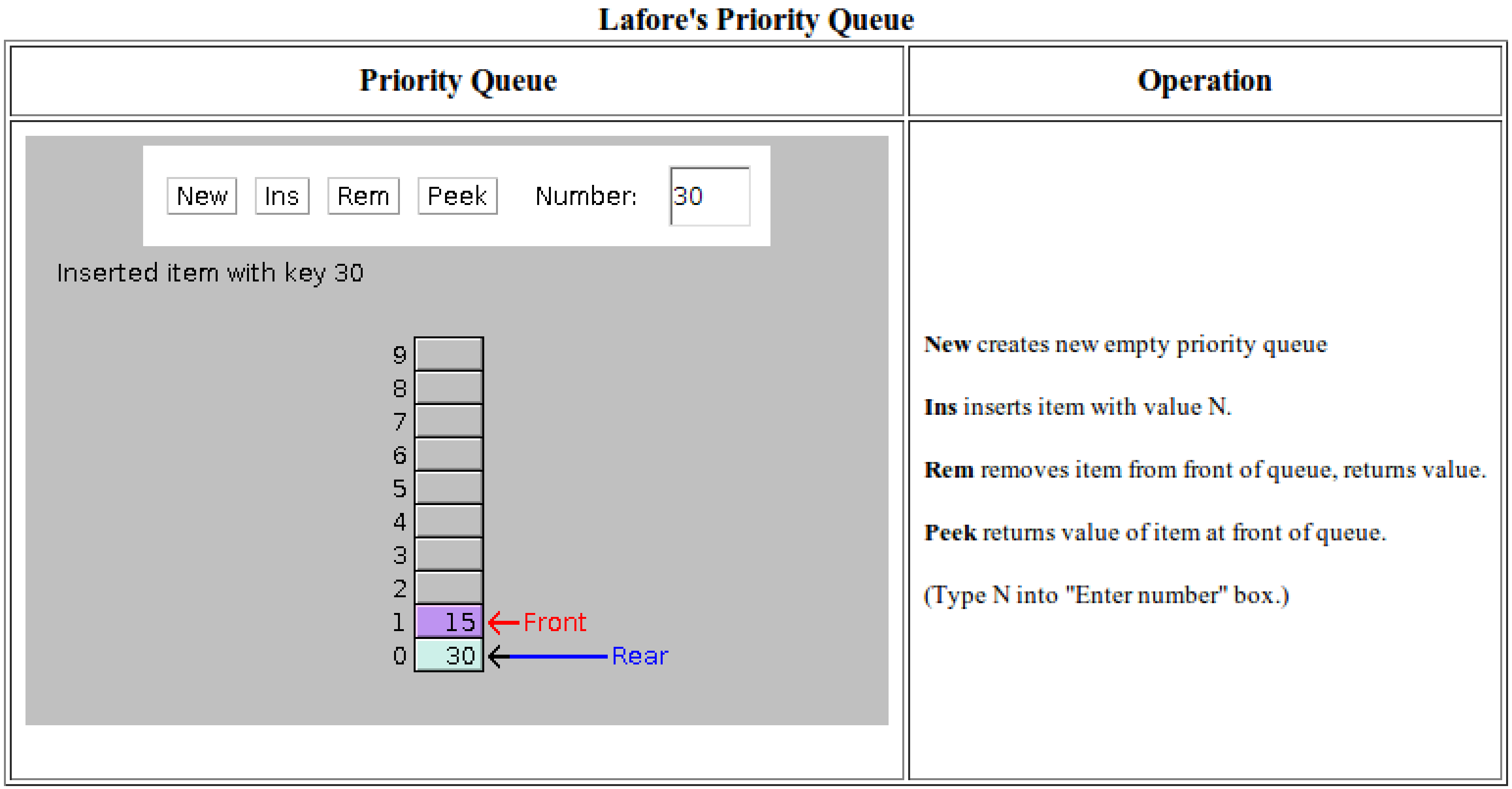
Insert
81

7	
6	
5	
4	
3	57
2	81
1	189
0	212

Insert
302

7	
6	
5	
4	57
3	81
2	189
1	212
0	302

Priority Queue Demo



Lafore's Priority Queue

Priority Queue	Operation
Number: 30 Inserted item with key 30	**New** creates new empty priority queue **Ins** inserts item with value N. **Rem** removes item from front of queue, returns value. **Peek** returns value of item at front of queue. (Type N into "Enter number" box.)

http://www2.latech.edu/~box/ds/PriorityQ/PriorityQ.html

PriorityQueue code

- Examples/Chap04/PriorityQ/priorityQ.java

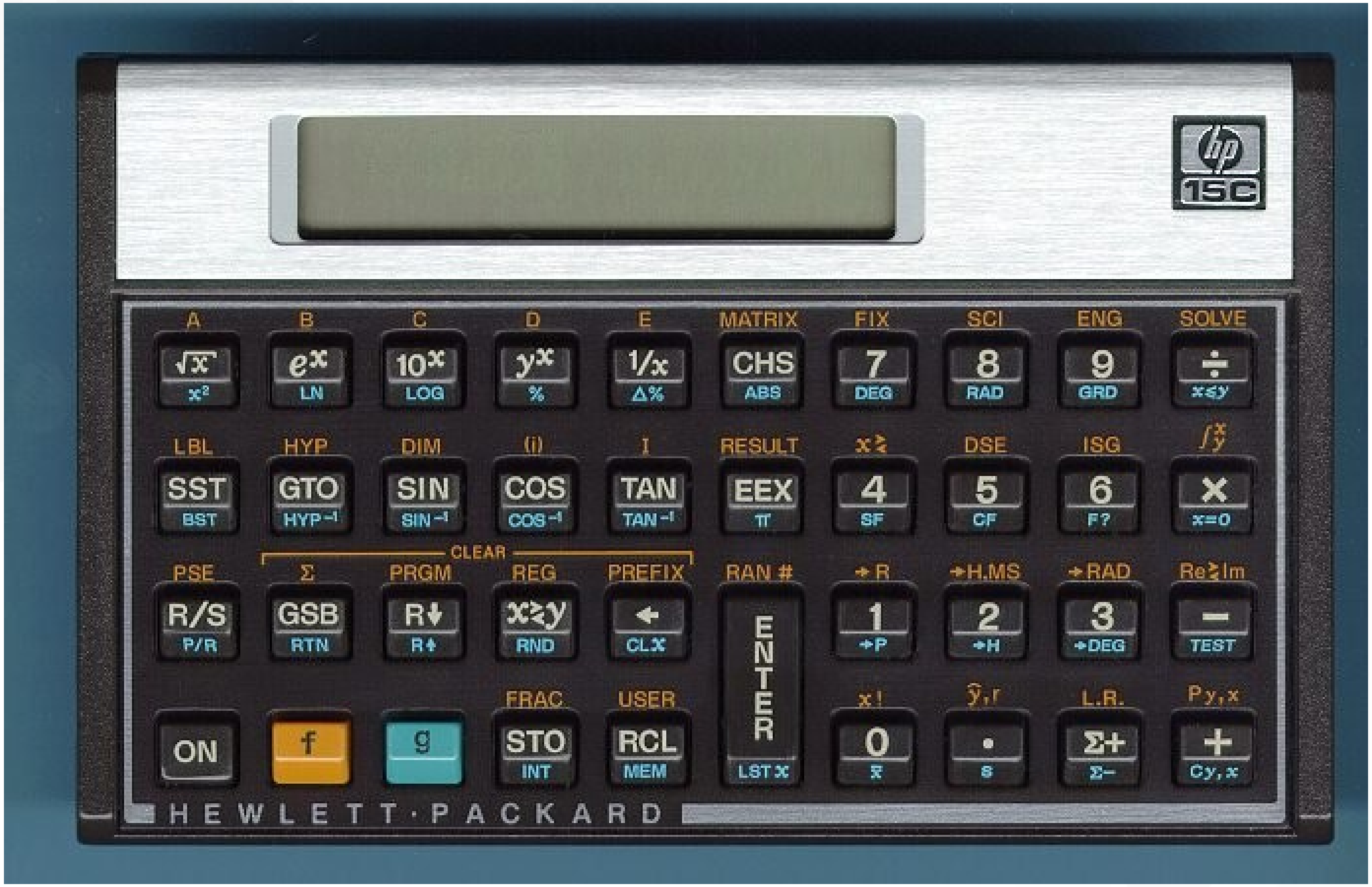
Efficiency of Priority Queue

- Our first implementation
  - Remove
    - No comparisons
    - No copying
    - One array access
    - O(1)
  - Insert
    - Find where to put it
    - Shift elements out of the way
    - Worst case requires shifting *n* elements
    - O(*n*) comparisons and copies

there is another way

we'll see later

Parsing Arithmetic Expressions



http://hp15c.com/

RPN

- Reverse Polish Notation
  - also known as postfix
- Instead of (3 + 4) do 3 4 +

  (3 + 11) + 5 → 3 11 + 5 +

  1 + 2 * 3 → 1 2 3 * +

  5 + (1 + 2) * 4 – 3 → 5 1 2 + 4 * + 3 -

http://en.wikipedia.org/wiki/Reverse_Polish_notation

Stack Operations

5 1 2 + 4 * + 3 -

Input	Operation	Stack	Comment
5	push	5	
1	push	5 1	
2	push	5 1 2	
+	add	5 3	pop two, add, push
4	push	5 3 4	
*	mult	5 12	pop two, mult, push
+	add	17	pop two, add, push
3	push	17 3	
-	sub	14	pop two, sub, push

Challenge

Convert the following expressions from infix to postfix

((2 + 4) * 7) + 3 * (9 – 5)

(1 + 1 + 2 + 1) * 3

2 * (1 + (2 * (1 + 1)))

Challenge Two

Convert the following postfix expressions back to infix

1 3 4 + * 2 *

2 4 + 7 * 3 9 5 - * +

1 1 1 1 1 + + + + 2 + 2 + 2 + 2 +

Parsing Infix

- Computing results of infix expression
  - Bit tricky
  - Requires scanning forward and back
  - Keep track of where we are, current state of parse
- Computing results of postfix expression
  - Just use a stack
  - Natural for a program

- Converting infix expressions to postfix requires parsing
  - Compilers do this all the time

Class and Variable Review

- Java Review part 1

Custom Class

```
class MyClass {
   public int value;
}
```

defining a custom class

```
MyClass o;
```

a variable with type of the custom class

```
o = new MyClass();
```

creating a new instance of the class

```
o.value = 5;
```

accessing public variables

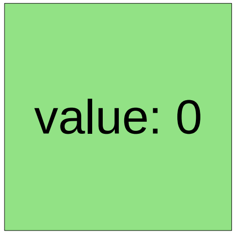
Names versus Values

```
class MyClass {
   public int value;
}


MyClass p, q;

p = new MyClass();
p.value = 42;

q = p;

q.value = 17;

What is p.value?
```

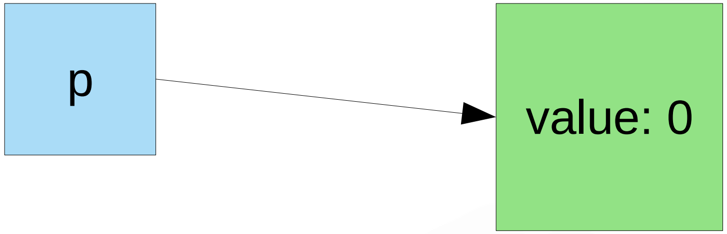
names versus values

Walkthrough

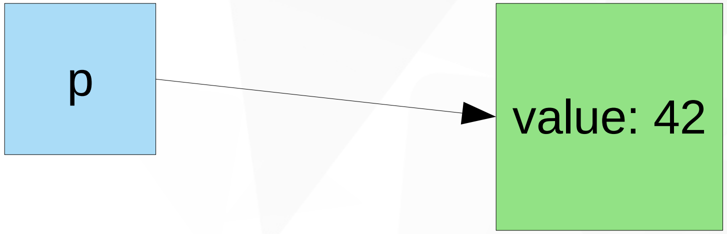
```
new MyClass()
```



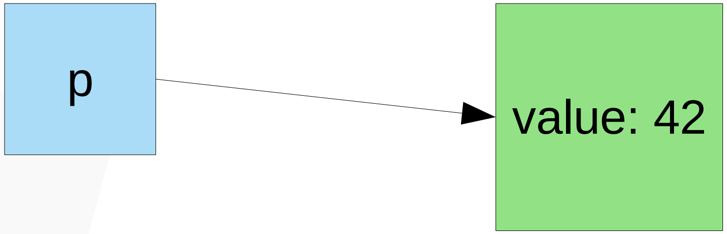
```
p = new MyClass()
```



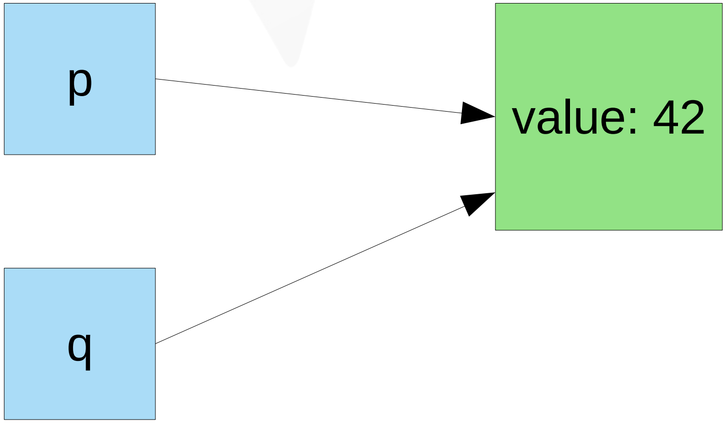
```
p.value = 42
```



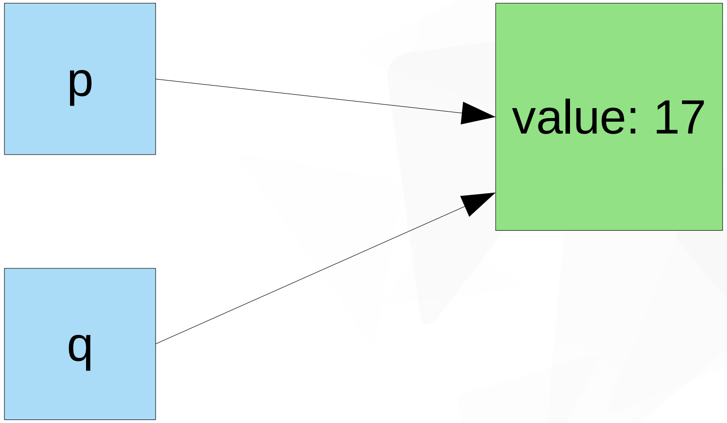
Walkthrough



```
q = p;
```



```
q.value = 17;
```



What happens?

```
class Pair {
   public int a;
   public String b;
}
```

```
Pair p, q, r;

p = new Pair();
p.a = 5;
p.b = “Time”;

q = p;

r = new Pair();
r.a = 12;
r.b = “Space”;

p.a = r.a;
p.b = r.b;

What is q.b?
```

Constructors

```
class Pair {
   private int damage;
   private String name;

   public Pair(int v) {
      damage = v * 10;
      name = “Phasers”;
   }
}
```

defining a constructor

```
Pair w = new Pair(12);
```

creating a new instance of the class

```
What is w.damage ?
```

Linked Lists

- "You can either have software quality or you can have pointer arithmetic, but you cannot have both at the same time."

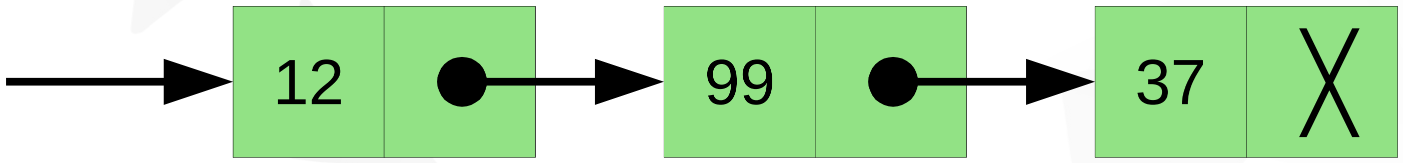
  Bertrand Meyer

What is a linked list?

- Linked lists are versatile data structure
  - 2nd most common after arrays/vectors
- Chain elements together with arrows going from one to the next
  - Can rearrange arrows to insert/delete anywhere in list
    - No shifting required 😄
  - Give up the ability to directly index into list ☹

Singly Linked List

- Three element list of integers
- Singly-linked
  - Each node has a value and a next node
  - Or next node is *null*, represented by X



Node

```
class Node {
    public int value;
    public Node next;
}
```

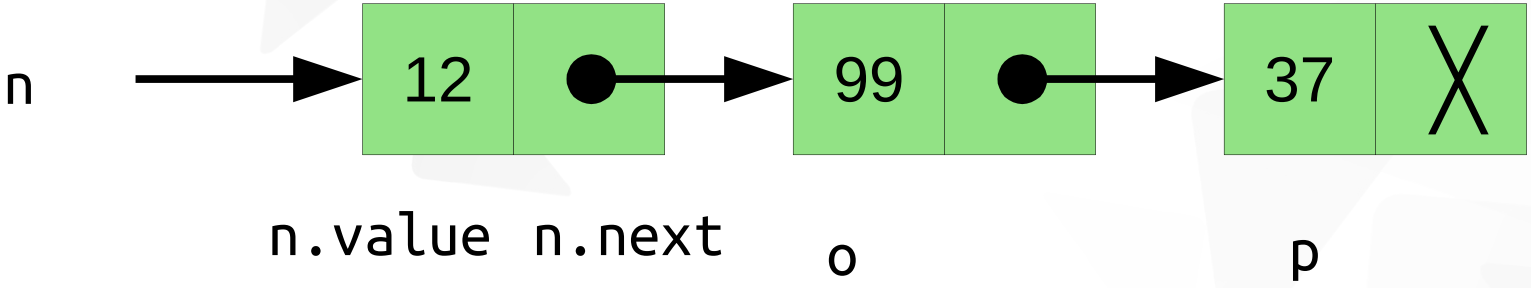
kind of like recursion

defining a class in terms of itself

Building a List

```
class Node {
   public int value;
   public Node next;
}
```

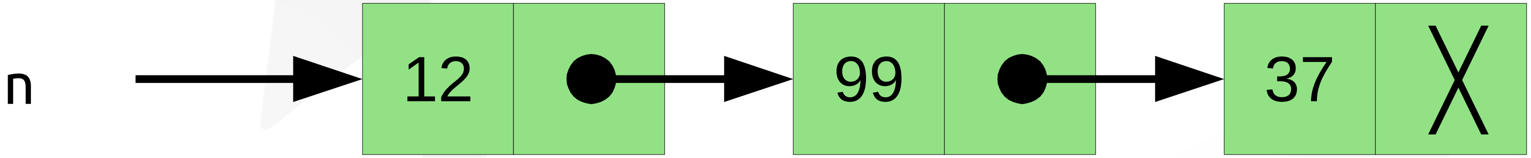
```
Node p = new Node();
p.value = 37;
p.next = null;
Node o = new Node();
o.value = 99;
o.next = p;
Node n = new Node();
n.value = 12;
n.next = o;
```



Building a List

```
class Node {
   public int value;
   public Node next;
}
```

```
Node n = new Node();
n.value = 12;
n.next = new Node();
n.next.value = 99;
n.next.next = new Node();
n.next.next.value = 37;
n.next.next.next = null;
```



n

12

99

37

n.value n.next

n.next.next

n.next.value

Linked List Class

```
class Node {
   public int value;
   public Node next;
}
```

```
class LinkList {
   private Node first;

   public LinkList() {
      first = null;
   }

   public void insertFirst(int value) {
      ...
   }
   .
   .
   .
}
```

Linked List Class - insertFirst

```
class Node {
    public int value;
    public Node next;
}
```

```
class LinkList {
    private Node first;

    public LinkList() {
        first = null;
    }

    public void insertFirst(int value) {
        ...
    }
    .
    .
    .
}
```

implement insertFirst

1. Draw node diagram for each line of test
2. Implement insertFirst

test

```
LinkList lst = new LinkList();
lst.insertFirst(37);
lst.insertFirst(99);
lst.insertFirst(12);
```

Linked List Class - deleteFirst

```
class Node {
    public int value;
    public Node next;
}
```

```
class LinkList {
    private Node first;

    public LinkList() {
        first = null;
    }

    public void deleteFirst() {
        ...
    }
    .
    .
    .
}
```

implement deleteFirst

1. Draw node diagram for each line of test

2. Implement deleteFirst

test

```
LinkList lst = new LinkList();
lst.insertFirst(12);
lst.insertFirst(5);
lst.insertFirst(8);
lst.deleteFirst();
lst.deleteFirst();
```

Linked List Class - find

```
class Node {
    public int value;
    public Node next;
}
```

implement find

```
class LinkList {
    private Node first;

    public LinkList() {
        first = null;
    }

    public Node find(int key) {
        ...
    }
    .
    .
    .
}
```

1. Implement find

2. Check against test

test

```
LinkList lst = new LinkList();
lst.insertFirst(17);
lst.insertFirst(90);
lst.insertFirst(22);
Node n90 = lst.find(90);
Node n17 = lst.find(17);
Node n23 = lst.find(23);
```

Linked List Class - insertLast

```
class Node {
    public int value;
    public Node next;
}
```

```
class LinkList {
    private Node first;

    public LinkList() {
        first = null;
    }

    public void insertLast(int value) {
        ...
    }
    .
    .
    .
}
```

implement insertLast

1. Draw node diagram for each line of test

2. Implement insertLast

test

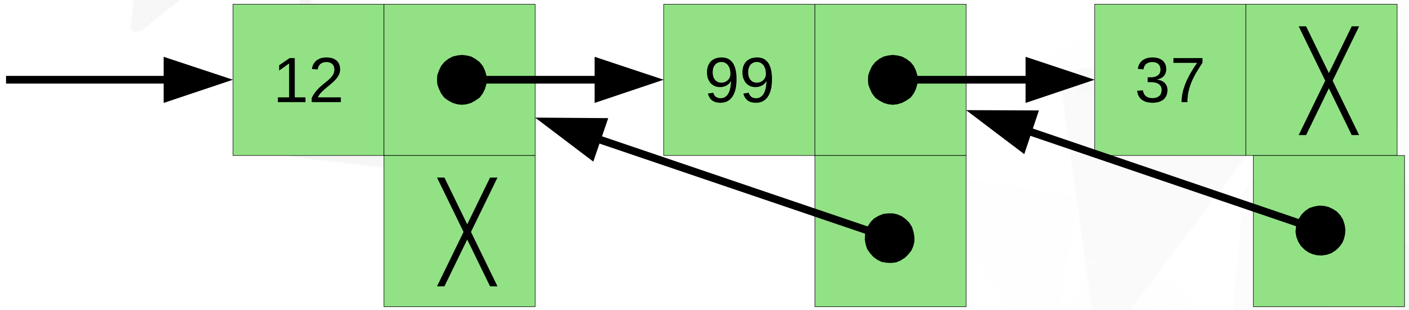
```
LinkList lst = new LinkList();
lst.insertFirst(100);
lst.insertLast(216);
lst.insertFirst(8);
lst.insertLast(3);
```

Methods of Linked Lists

- *isEmpty* - Check if list is empty
- *insertFirst* - Insert new value at start of list
- *insertLast* - Insert new value at end of list
- *deleteFirst* - Delete value at start of list
- *find* - Find a given value in the list
- *delete* - Delete a value somewhere in the list

Doubly-Linked Lists

- Wouldn't it be nice to be able to go forwards and backwards any time we wanted?



Doubly-Linked List Node

```
class Node {
   public int value;
   public Node next;
   public Node prev;
}
```

The End