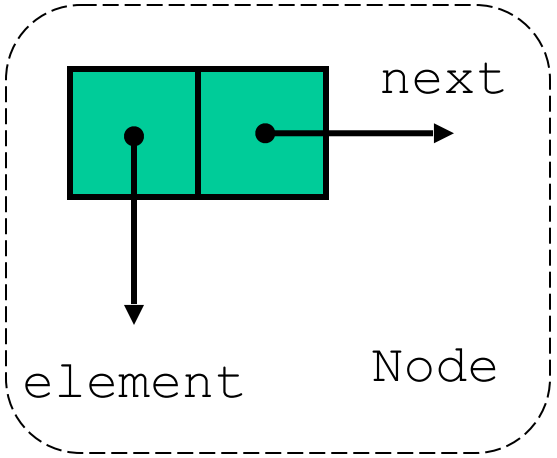
CS206

Generic Linked Lists

Generic Linked List

- A linked list not locked into one type of object




```
private class Node<E> {
   public E element;
   public Node<E> next;
   public Node(E element, Node<E>
next) {
        this.element = element;
        this.next = next;
     }
```

Basics

```
private Node<E> head = null;
private Node<E> tail = null;
private int size = 0;
public int size() {return size;}
public boolean isEmpty() {return size == 0;}

public E first() {
  if (isEmpty()) {
     return null;
 }
  else {
      return head.element;}
}
```

Insertion & Deletion

```
 public void addLast(E e) {
   Node<E> newest = new Node<>(e, null);
   if (isEmpty()) { head = newest;}
   else {tail.next=newest;}
   tail = newest;
   size++;
 }

public E removeFirst() {
   if (isEmpty()) {return null;}
   E target = head.element;
   head = head.next;
   size--;
   if (isEmpty()) {tail = null;}
   return target;
 }
```

Find

Recall from RabbitLinkedList

Problem

```
public Rabbit find(String id)
{
    Node curr = head;
    while (curr!=null)
    {
        if (curr.data.getId().equals(id))
        {
       return curr.data;
        }
        curr=curr.next;
    }
    return null;
}
```

Need a generic way to compare!!!

The Comparable Interface

```
public interface Comparable<T>
  72: {
  73:   /**
  74:    * Compares this object with another, and returns a numerical result based
  75:    * on the comparison.  If the result is negative, this object sorts less
  76:    * than the other; if 0, the two are equal, and if positive, this object
  77:    * sorts greater than the other.  To translate this into boolean, simply
  78:    * perform <code>o1.compareTo(o2) <em>&lt;op&gt;</em> 0</code>, where op
  79:    * is one of &lt;, &lt;=, =, !=, &gt;, or &gt;=.
  80:    *
  81:    * (deleted more)    *
  92:    * @param o the object to be compared
  93:    * @return an integer describing the comparison
  94:    * @throws NullPointerException if o is null
  95:    * @throws ClassCastException if o cannot be compared
  96:    */
  97:   int compareTo(T o);
  98: }
```

Short story: return 0 if equal, negative if less, positive if greater

Comparable example
Integer and String

```
public class Main {

   public static void main(String[] args) {
      Integer i5 = new Integer(50);
      Integer i3 = new Integer(30);
      Integer j5 = new Integer(50);
      System.out.println("3 c 5 " + i3.compareTo(i5));
      System.out.println("5 c 3 " + i5.compareTo(i3));
      System.out.println("5 c 5 " + i5.compareTo(j5));
      System.out.println("5 eq 5 " + (i5 == j5));

      Integer k5 = 5;
      Integer l5 = 5;
      System.out.println("5 eq 5 " + (k5 == l5));

      String abc = "abc";
      String def = "def";
      String abc0 = new String("abc");
      System.out.println("a c d " + abc.compareTo(def));
      System.out.println("5 c 3 " + def.compareTo(abc));
      System.out.println("5 c 5 " + abc.compareTo(abc0));
   }
}
```

Generic with Comparable

```
public interface LinkedListInterface<E extends Comparable<E>>
{ // otherwise unchanged
}

public class LinkedList<E extends Comparable<E>> implements
LinkedListInterface<E> {
    private class Node<E>
    {
       public Comparable<E> data;
       public Node next;
       public Node(Comparable<E> data, Node next)
       {
          this.data = data;
          this.next = next;
   }  }
```

With these changes this linked list class requires that any class being stored within implement the comparable interface

Find with Comparable

```
@Override
public E find(E e)
{
    Node curr = head;
    while (curr!=null)
    {
        if (curr.data.compareTo(e)==0)
        {
            return (E)curr.data;
        }
        curr=curr.next;
    }
    return null;
}
```

Comparable Rabbit

```
public class Rabbit implements Comparable<Rabbit>
{
    // otherwise unchanged
    @Override
     public int compareTo(Rabbit r)
     {
         return iD.compareTo(r.getId());
     }
}
```

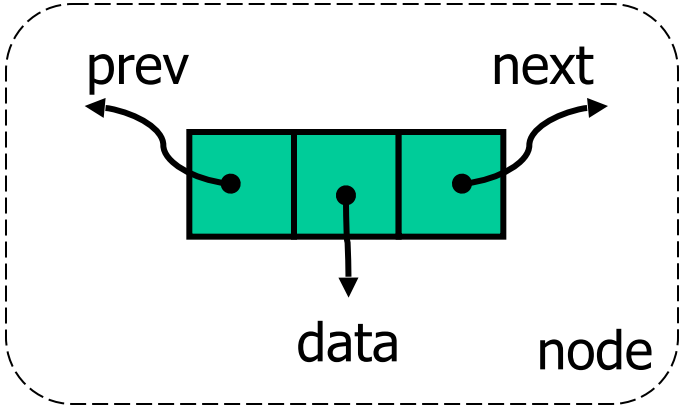
Goto unix and run

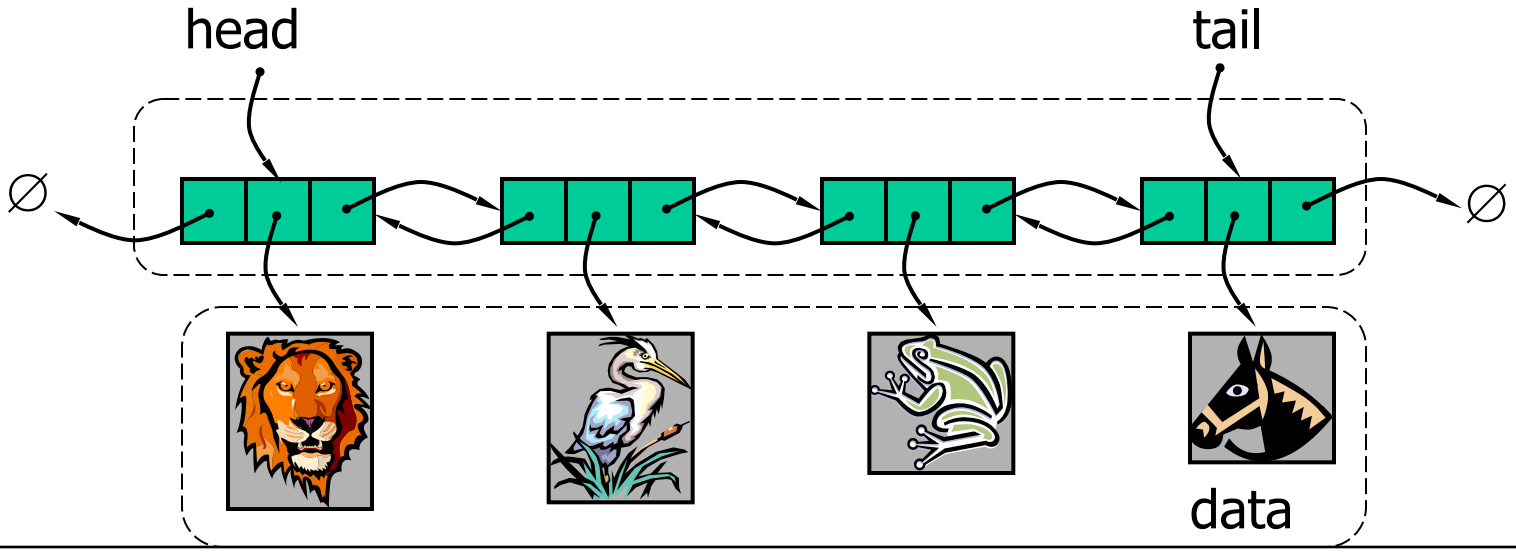
CS206

Doubly and Circular Linked Lists

Doubly Linked List

- Can be traversed forward and backward
- Nodes store an extra reference





Rabbity Double Linked List interface

```
public interface DoubleLinkedListInterface
{
    int size();
    boolean isEmpty();
    Rabbit first();
    Rabbit last();
    void addLast(Rabbit c);
    void addFirst(Rabbit c);
    Rabbit removeFirst();
    Rabbit removeLast();
    Rabbit remove(Rabbit r);
    Rabbit find(String iD);
}
```

Other than name, this is identical to single linked list!!!

Node

```
public class RabbitDLL {
   private class Node {
     public Rabbit data;
     public Node prev, next;
     public Node(Rabbit data, Node prev, Node next)
   {
       this.data = data;
       this.prev = prev; this.next = next;
     }
    }
```

Basics

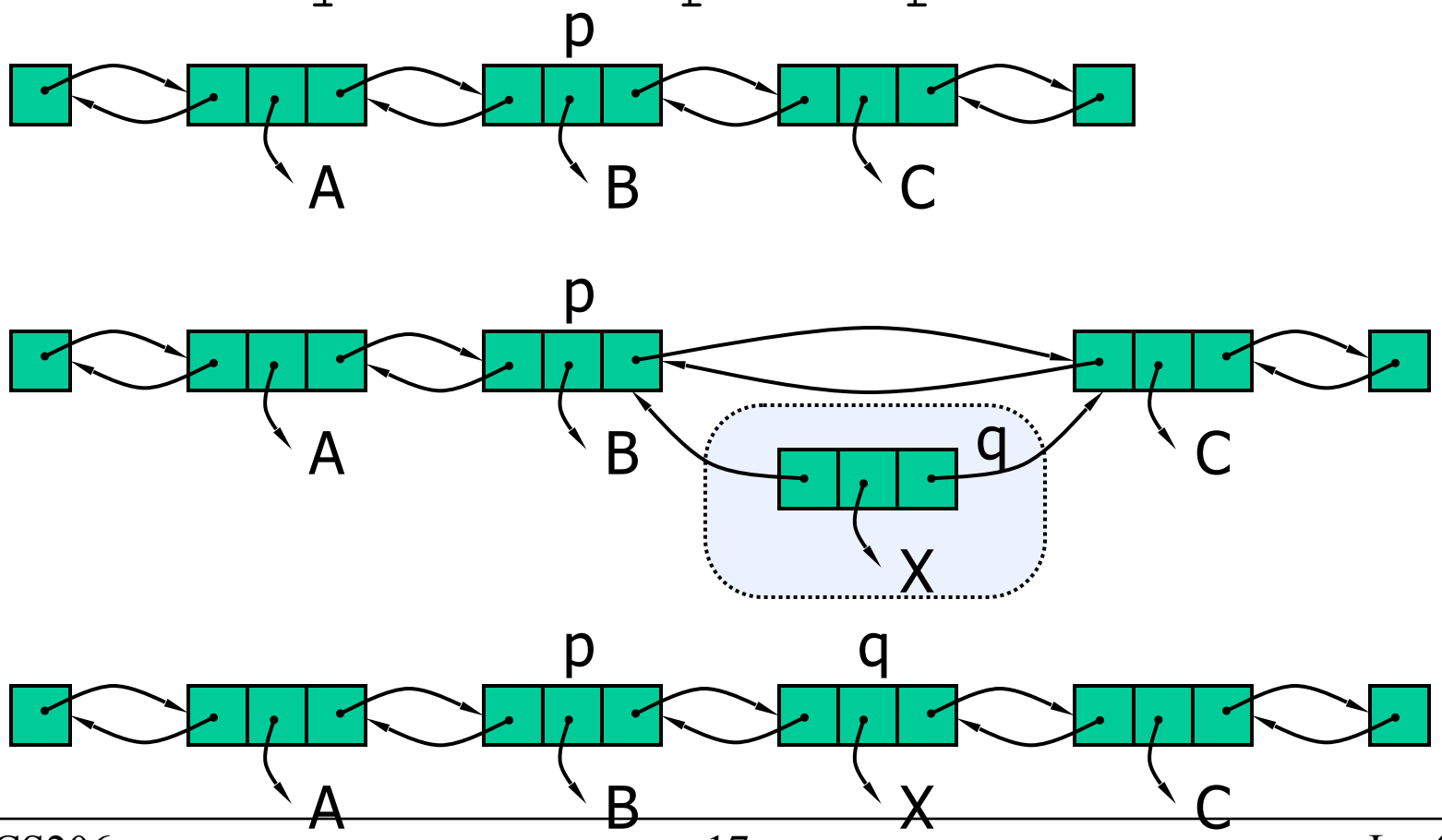
```
private Node head = null;
private Node tail = null;
private int size = 0;
public int size() {return size;}
public boolean isEmpty() {return size == 0;}
public Rabbit first() {
  if (isEmpty()) {return null;}
  else {return head.data;}
}
```

Insertion

```
public void addLast(Rabbit c) {
  Node newest = new Node(c, tail, null);
  if (isEmpty()) { head = tail = newest;}
  else {
    tail.next = newest;
    tail = newest;
  }
  size++;
}
```

Add Between

- Insert `q` between `p` and `p.next`



Add Between

```
public void addBtw(Rabbit c, Node prev, Node next)
{
    Node newest = new Node(c, prev, next);
    prev.next = newest;
    next.prev = newest;
    size++;
}
```

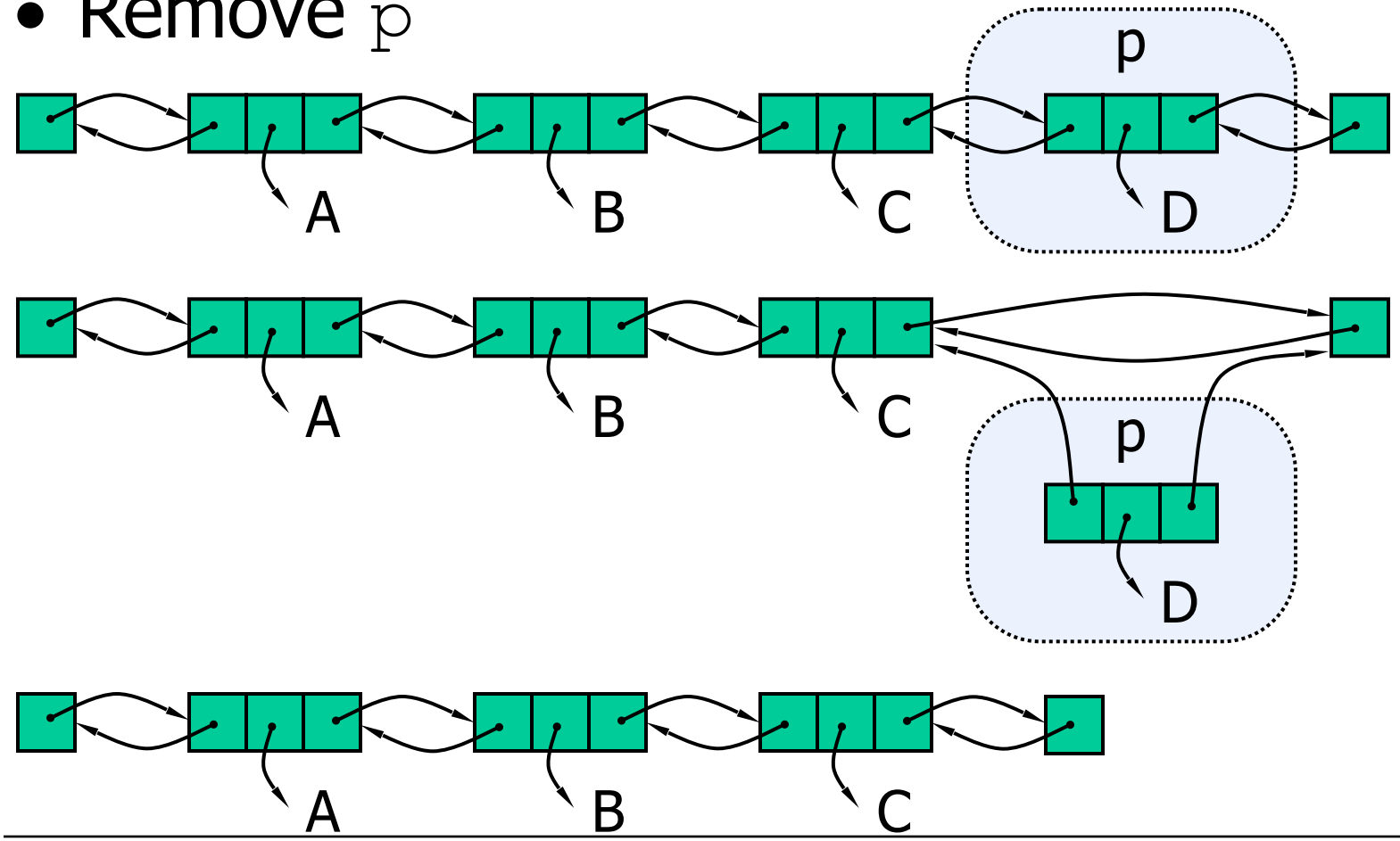
Problems??

Deletion

```
public Rabbit removeFirst() {
  if (isEmpty()) {return null;}
  Node target = head;
  if (head == tail) {
    head = tail = null;
  }
  else {
    head = head.next; head.prev=null;
  }
  size--;
  return target.data;
}
```

Deletion

- Remove `p`



Deletion

```
public void City remove(Rabbit r)
{
  Node n = findNode(r);
  if (head == n) {removeFirst();}
  else if (tail == n) {removeLast();}
  else {
    n.prev.next=n.next;
    n.next.prev=n.prev;
    size--;
    return n.data;
  }
}
```

Circular Linked List

- when order doesn't matter