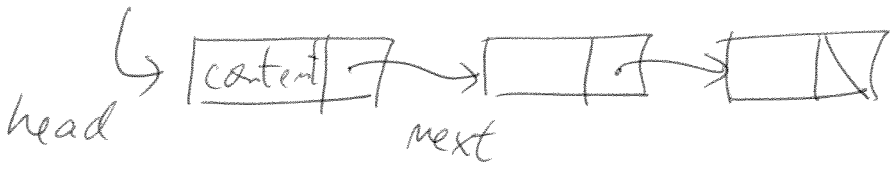
COMP 250 - Lecture 20

Last time: Single-linked lists



Insertion may be O(1) or O(n) (first or last)

Finding an element (linear search)

```
// Inside the SingleLinkedList class
public boolean find (Object c){
 if (head == null) return false;
    //isEmpty()
 //use crt as an Iterator (points to current place)
NodeCrt = head;
 while (crt != null) {          == only for primitive types
    if c.equals(crt.content) return true;
    crt = crt.next; // go to next element
 }
 return false;
}
```

Linear processing in general:

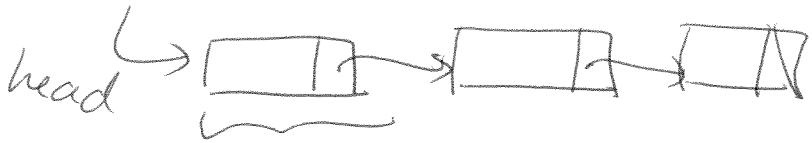
```
Node crt = head;
while (crt != null) {
    // do something to content
    crt = crt.next;
}
```

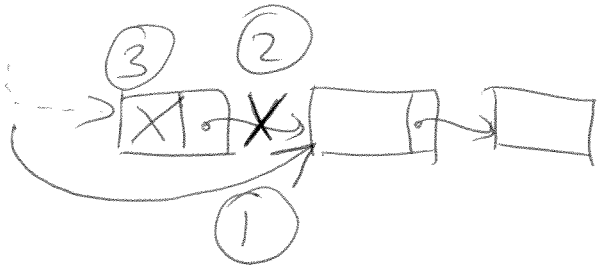
Alternatively:

```
for (Node crt = head; crt != null; crt = crt.next)
    // do something to crt.content
```

O(n)

Removing elements

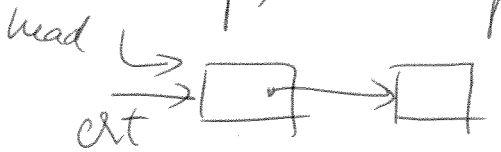


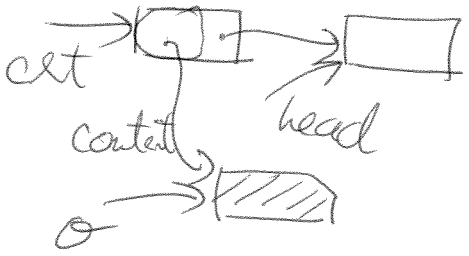


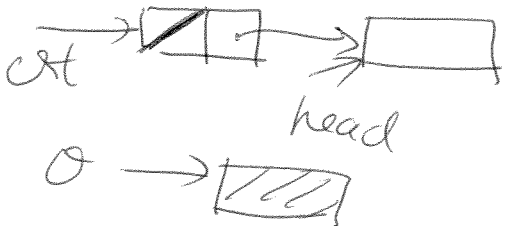

```
Node crt = head;
head = crt.next
crt.next = null;
crt.content = null; // to allow GC to take
                    content that is no longer needed
```

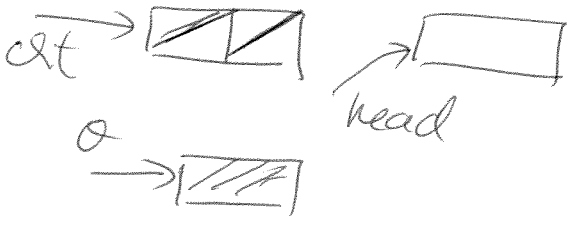
L20

```
public Object removeFirst() throws EmptyListException {
    if (head == null) throw new EmptyListException();
    Node crt = head;
    head = head.next;
    Object o = crt.content;
    crt.content = null;
    crt.next = null;
    return o;
}
```



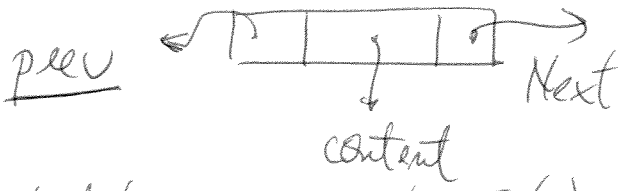






Double-Linked Lists

Node

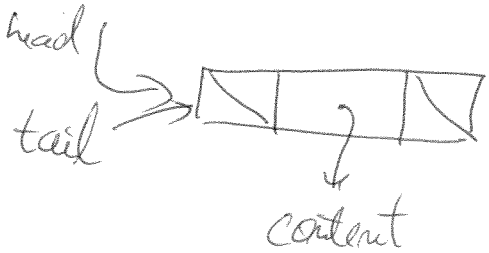


More flexibility; though O() same; more memory

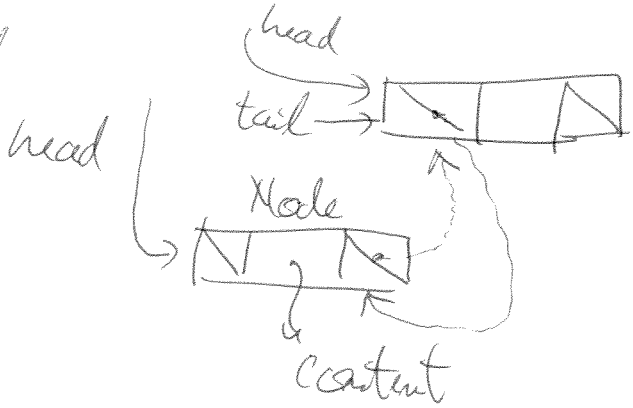
```
public class Node {
    Object content;
    Node next, prev; // initialized to null
    // constructor...
}

public class DoubleLinkedList {
    Node head, tail; // beginning & end of list
    //Insert in first position in a list
```

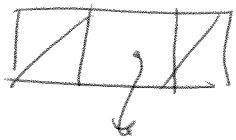
(1) head = tail = null (empty list)

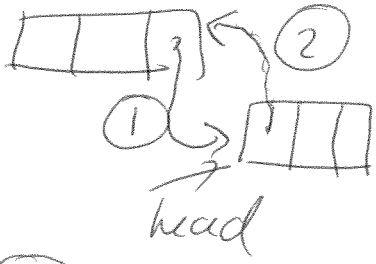


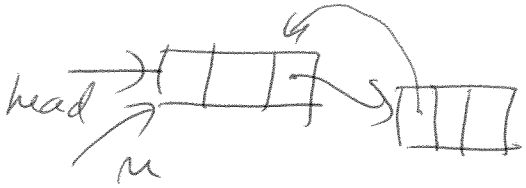
(2) head = tail ≠ null



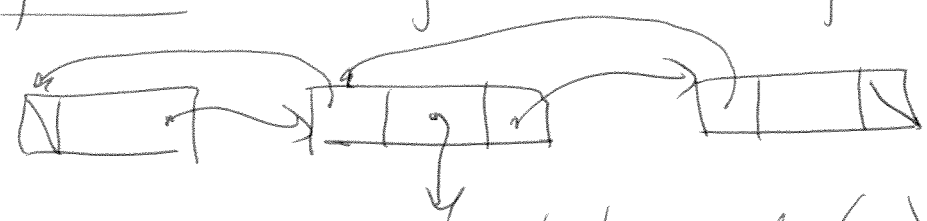
```
public void insertFirst(Object c) {
  Node n = new Node(c);
  // Link n at front of our list
  if head
  n.next = head;   (1)
  if (head != null) head.prev = n;   (2)
  head = n;
// might need to adjust tail
  if (tail == null) // we had empty list before
      tail = n;
}
```





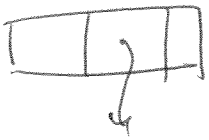


insertAfter Object c Object o



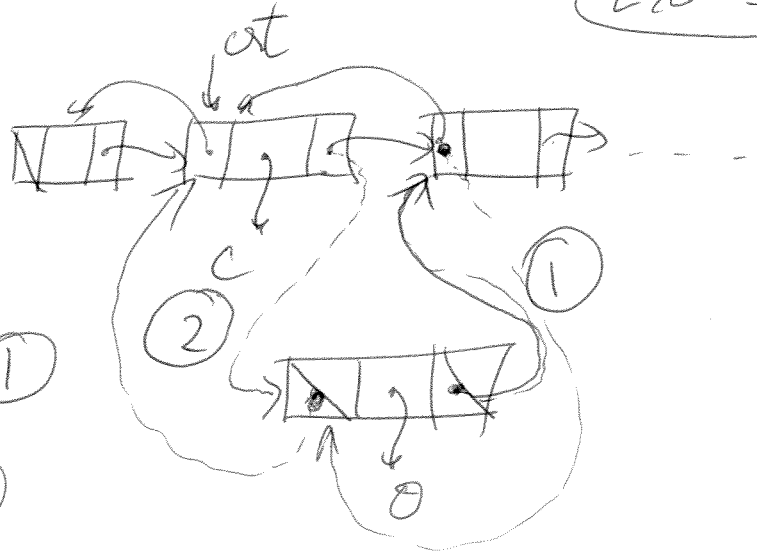
content.equals(c)

insert new object after this



```
public void insertAfter(Object c, Object o) {   throws Exception
// Look for c in the list
Node crt = head;
while (crt != null) {
        if ((crt.content).equals(c)) break;
        crt = crt.next;
}
// check how we terminated
if (crt == null) throw new Exception();
```

// we have content




```
Node newNode = new Node(o);
newNode.next = crt.next;  (1)
crt.next = newNode;  (2)
newNode.prev = crt;
if (newNode.next != null)
    newNode.next.prev = newNode;
// adjust head or tail of list if need be
//   head: cannot change;  tail: might need to change
if (crt == tail)          // == reference comparison
    tail = newNode;       // not content comparison
```