Linked List I

Reminder

- Project 1
  - If not already picked up, do so after class.
  - Still have some Exam 1's left
- Project 2
  - Due this Sunday
    - Submit early!
    - Submit often!
  - Miss the minimum? Please see me after class.
  - Questions?

Exam 2

- Will return and review tomorrow.

Announcement

- Final Exam
  - Wednesday, February 25, 2004
  - 8:00am – 10:00 am
  - 70-3435

Any questions

Before we start linked lists

- As promised
  - The hashing applet.

Plan for today

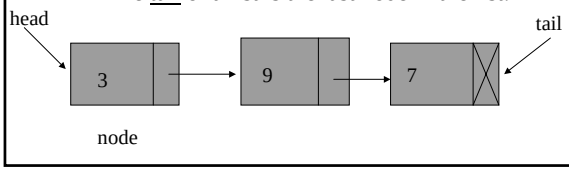
- Introduction to linked list
- Testing tips

Linked Lists

- Sequence of elements
  - arranged one after the other
  - Each element has
    - Some piece of data
    - a link to the next element in the sequence
    - The “next” link for the last element is null.
  - Basic linked list need not be sorted.

Linked Lists

- Implementation
  - Like with trees, a list can be seen as a collection of “nodes”
  - The head of a list is the first node in the list
  - The tail of a list is the last node in the list.



Linked Lists

- Implementation

```
public class ListNode {
     Object data;
     ListNode next;
}
```

ListNodes

- Operations on ListNodes
  - Constructor
    - initialData – data to be placed in the node
    - initialNext – reference to the node that is after this node in the list
  - Get Methods
    - Object getData()
    - ListNode getNext()
  - Set Methods
    - setData(Object O)
    - setNext (ListNode N)

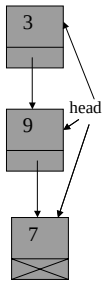
Linked List

- Let’s create a simple list:

```
// Add the 7
ListNode head = new ListNode (new Integer(7), null);

// Add the 9
head = new ListNode (new Integer(9), head);

// Add the 3
head = new ListNode (new Integer(3), head);
```



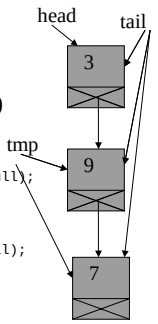
Linked List

- Let's create a simple list (another way)

```
// Add the 3
ListNode head = new ListNode (new Integer(3), null);
ListNode tail = head;

// Add the 9
ListNode tmp = new ListNode (new Integer(9), null);
tail.setNext (tmp);
tail = tmp;

// Add the 7
tmp = new ListNode (new Integer(7), null);
tail.setNext (tmp);
tail = tmp;
```

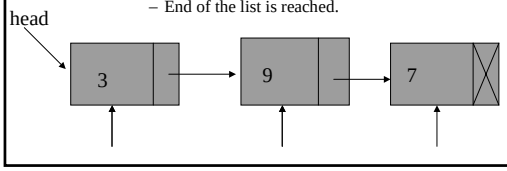


Linked List

- Operations on entire list
  - Find
  - Add
  - Remove

Linked List – Find

- Finding an item in a linked list
  - Basic idea
    - Start at head of the list
    - Follow the links until
      - Object searched for is found or
      - End of the list is reached.



Linked List – Find

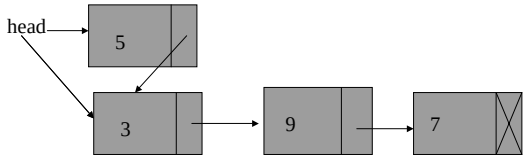
```
public ListNode find (ListNode head, Object
  target)
{
   ListNode found = null;
   ListNode cur = head;
   while ((found == null) && (cur != null)) {
      if (target.equals (cur.getData())
         found = cur;
      else
         cur = cur.getNext();
   }
   return found
}
```

Linked List – Add

- Adding will depend on where you wish to add the new node
  - Add before the head of the list
  - Add to the interior of the list

Linked List – Add

- Adding before the head of the list
  - Make the new node the new head



Linked List – Add

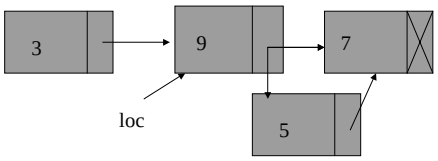
```
// Add the 7
ListNode head = new ListNode (new Integer(7), null);

// Add the 9
head = new ListNode (new Integer(9), head);

// Add the 3
head = new ListNode (new Integer(3), head);
```

Linked List – Add

- Adding to the interior of the list
  - Need a reference to the node before the location of the new node (loc)
  - New node's next will point to whatever loc's next is pointing to
  - loc.next will point to new node



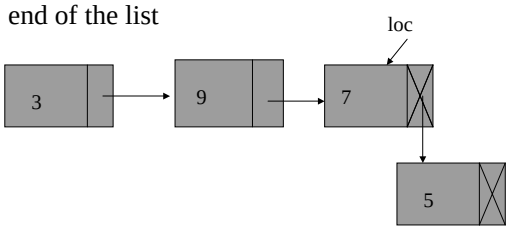
Linked List – Add

```
public void addAfter (ListNode loc,
  Object target)
{
  ListNode tmp = new ListNode (target,
                               loc.getNext();

  loc.setNext(tmp);
}
```

Linked List – Add

- Note that this also works if adding to the end of the list

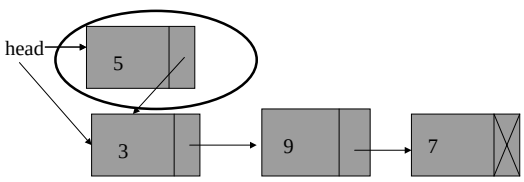


Linked List – Remove

- Remove will depend on where the node you wish to remove is
  - Removing the node at the head of the list
  - Removing an interior node.

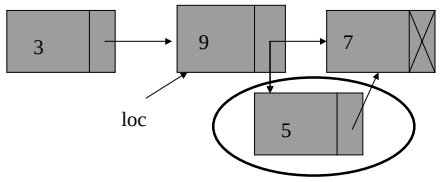
Linked List – Remove

- Removing the node at the head of the list
  - Have the head point to the node after the old head
    - `head = head.getNext();`



Linked List – Remove

- Removing an interior node
  - Need a reference to the node before the node to be deleted (loc)
  - Loc's next will point to whatever the deleted node's next was pointing to.



Linked List – Remove

```
public void removeAfter (ListNode loc)
{
  ListNode del = loc.getNext();
  if (del != null)
     loc.setNext(del.getNext());
}
```

Summary

- Linked Lists
- ListNode
- Operations
  - Add
  - Find
  - Remove

Linked List – Remove

- Remove is a bit awkward
  - Suppose we want to remove the node returned by `find`.
    - We'll need to maintain a pointer to the node before this node.
    - Very cumbersome.
    - Possible solution
      - Have a node have a pointer to it's previous element in the list as well.
      - Doubly-linked list