Linked List III

A Linked List Class

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?

Announcement

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

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.

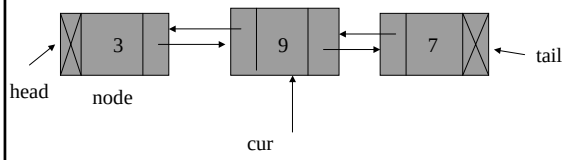
Doubly Linked Lists

- Sequence of elements
  - 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.
    - a link to the previous element in the sequence
      - The "prev" link for the first element is null.

A Linked List class

- Let's encapsulate all this into a single class:
  - LinkedList
    - A class that implements a linked list
    - head – points to the head of the list
    - tail – points to the end of the list
    - cur – points to the "current" element in the list.
      - We move this cur pointer to iterate through the list.

A Linked List class



A Linked List class

- Operations:
  - Add / Remove
    - add – add an object after the “current” element
    - insert – add an object before the “current” element
    - remove – remove the current element
  - Search
    - find – find an element in the list and set it to be the “current” element.

A Linked List class

- Operations:
  - Iteration
    - firstElem – returns the first element of the list and sets the “current” element to the first in the list
    - lastElem -- returns the last element of the list and sets the “current” element to the last in the list
    - nextElem – Moves the “current” element forward and returns it
    - prevElem – Moves the “current” element backwards and returns it.
  - All will return null if there is no current element.

A Linked List class

```
public class LinkedList {
  DListNode head;
  DListNode tail;
  DListNode cur;

  public LinkedList()
  {
      head = tail = cur = null;
  }
}
```

A Linked List class

- Add
  - Adds an element after the current element
  - Sets the current element to be the one just added.

```
L.add (new Integer(1));
L.add (new Integer(2));
L.add (new Integer(3));
L.add (new Integer(4));
L.add (new Integer(5));
```

1 2 3 4 5

A Linked List class

- Add

```
public void add (Object O) throws ListException
{
   if (head == null) {
        head = new DListNode (O, null, null);
        tail = cur = head;
   }
   else {
        if (cur == null)
            throw new ListException ("No current element");

        DListNode tmp = new DListNode (O, cur.getNext(), cur);
        if (cur == tail) tail = tmp;
        cur = tmp;
   }
}
```

A Linked List class

- Insert
  - Inserts an element before the current element
  - Sets the current element to be the element just added.

```
L.insert (new Integer(1));
L.insert (new Integer(2));
L.insert (new Integer(3));
L.insert (new Integer(4));
L.insert (new Integer(5));
5 4 3 2 1
```

A Linked List class

- Insert

```
public void insert (Object O) throws ListException
{
  if (head == null) {
       head = new DListNode (O, null, null);
       tail = cur = head;
  }
  else {
       if (cur == null) throw new ListException ("No current
  element");

       DListNode tmp;
       if (cur == head) {
               head = new DListNode (O, head, null );
               cur = head;
       }
       else {
               tmp = new DListNode (O, cur, cur.getPrev());
               cur = tmp;
       }
```

A Linked List class

- Remove
  - Removes the current element
  - Sets the current element to be the element after the deleted element.

```
5 4 3 2 1
L.remove();
L.remove();
3 2 1
```

A Linked List class

- Remove

```
public void remove() throws ListException
{
   if (cur == null) throw new ListException ("No current element");

   DListNode nxt = cur.getNext();
   DListNode prv = cur.getPrev();
   if (cur == head) {
       head = head.getNext();
       head.setPrev (null);
   }
   else {
       prv.setNext(nxt);
   }

   if (cur == tail) tail = prv;
   cur = nxt;
}
```

A Linked List class

- Find
  - Searches for an element. If found returns true and sets current element to the element just found.
  - If not found, returns false and current element is null.

```
5 4 3 2 1
L.find (new Integer(3));
L.add (new Integer (33));
```

5 4 3 33 2 1

A linked list class

- Find

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

A linked list class

- FirstElem
  - Returns first element
  - Sets current to the first element
- LastElem
  - Returns last element
  - Sets current to the last element
- NextElem
  - Returns next element (null if at end of list)
  - Sets current to the next element
- PrevElem
  - Returns previous element (null if at beginning of list)
  - Sets current to the previous element

A linked list class

- To iterate through a list:

```
// print forward
Object D;
for (D=L.firstElem(); D != null; D =
  L.nextElem())
  System.out.print (D + " ");
System.out.println();

// print backwards
for (D=L.lastElem(); D != null; D = L.prevElem())
  System.out.print (D + " ");
System.out.println();
```

A Linked List class

- FirstElem

```
public Object firstElem()
{
  cur = head;
  if (cur == null) return null;

  return cur.getData();
}
```

A Linked List class

- LastElem

```
public Object lastElem()
{
  cur = tail;
  if (cur == null) return null;

  return cur.getData();
}
```

A Linked List class

- NextElem

```
public Object nextElem()
{
  if ((cur == null) || (cur == tail))
      return null;

  cur = cur.getNext();
  return cur.getData();
}
```

A Linked List class

- PrevElem

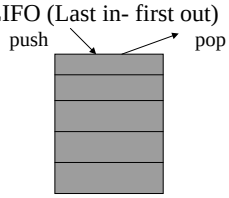
```
public Object prevElem()
{
  if ((cur == null) || (cur == head))
      return null;

  cur = cur.getPrev();
  return cur.getData();

}
```

A Linked List class

- Questions?

What can we do with a linked list?

- Let's build a stack
  - LIFO (Last in- first out)



Let's build a stack

- A stack can be implemented using a linked list:
  - Always insert an element "pushed" at the beginning of the list
  - Always pop an element from the beginning of the list.

Let's build a stack

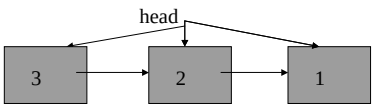
```
public Lstack {
     LinkedList data;
}
```

Let's build a stack

- Stack Operations
  - Push
  - Pop
  - Print
  - isEmpty

Let's build a stack

- Push
  - Insert item at the beginning of the list
    - Push 1
    - Push 2
    - Push 3

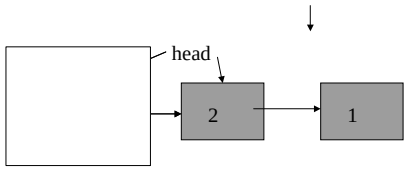


Let's Build a Stack

```
public void push (Object O)
{
    data.firstElem();
    try {
        data.insert (O);
    }
    catch (ListException E) {}
}
```

Let's Build a Stack

- Pop
  - Remove the first element of the list and return it

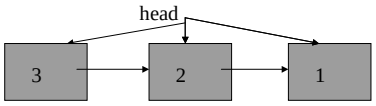


Let's Build a Stack

```
public Object pop()
{
  Object O = data.firstElem();
  try {
     if (O != null) data.remove();
     else System.out.println
        ("Trying to pop an empty stack");
  }
  catch (ListException E) {}
  return O;
}
```

Let's build a stack

- Print
  - Iterate through the elements from from to back and print each one



Let's build a stack

```
public void print ()
{
      Object O;
      for (O = data.firstElem(); O != null;
           O = data.nextElem()) {
            System.out.print (O);
      }
}
```

Let's build a stack

- isEmpty()
  - The stack is empty if the list has no first element.

Let’s build a stack

```
public boolean isEmpty()
  {
     Object O = data.firstElem();
     return (O == null);
  }
```

Summary

- Classes:
  - LinkedList
  - Lstack

Next week

- Last week of class
- Final exam: Wednesday, February 25
- M, T: Ownership & Intellectual Property
- W: Final exam review / course evals