CS 61B Data Structures and Programming Methodology

June 26 2008
David Sun
Announcements

• Project 1 will be handed out at the end of next week.
• Project 1 is now due after Midterm I on 7/15.
• Lab2 homework is due next Wed.
• George has extra space in his labs (5-7p.m) and discussions (5-6p.m).
• Another reminder, please send your emails to cs61b@imail.eecs.berkeley.edu
Yesterday...

• **Key features of Arrays:**
  – Stores a sequence of variables in continuous memory.
  – Array elements are indexed by a variable.
  – This allows for fast *random access/lookup*, e.g.,
    ```
    char[] c = {'b', 'l', 'u', 'e'};
    c = 'i';
    char x = c[3]; //to get element 3
    ```

• **What about *insertion*, say the character ‘i’ position 3?**
  – Arrays have fixed length that can’t be changed!
  – Cost of *Insertion* is proportional to the *length* of array.
Lists
Array Implementation

```java
public class List {
    int[] a;
    int lastItem;

    public List() {
        a = new int[10];
        lastItem = -1;
    }

    public void insertItem(int newItem, int location) {
        int i;
        for (i = lastItem; i >= location; i--) {
            a[i + 1] = a[i];
        }
        a[location] = newItem;
        lastItem++;
    }
}
```

Shift the items to the right by one position.
ArrayList

public void insertItem(int newItem, int location) {
    int i;

    if (lastItem + 1 == a.length) {
        int[] b = new int[2 * a.length];
        for (i = 0; i <= lastItem; i++) {
            b[i] = a[i];
        }
        a = b;
    }

    for (i = lastItem; i >= location; i--) {
        a[i + 1] = a[i];
    }

    a[location] = newItem;
    lastItem++;
}

creates a new array twice as long and copy the elements of the small array into the larger array.

could also use System.arraycopy(..)
Linked List

• A linked list is made up of *nodes*; a node stores:
  – an item
  – an reference to next node in the list

```java
public class ListNode {
    int item;
    ListNode next;
}
```

• Recursive definition: the definition of the class refers to itself.
An Example

ListNode l1 = new ListNode();
ListNode l2 = new ListNode();
ListNode l3 = new ListNode();

l1.item = 1;
l2.item = 2;
l3.item = 3;

l1.next = l2;
l2.next = l3;
l3.next = null;  // good practice to initialize
                 // explicitly
ListNode Constructors

```java
public ListNode(int item, ListNode next) {
    this.item = item;
    this.next = next;
}

public ListNode(int item) {
    this(item, null);
}

ListNode l1 = new ListNode(1, new ListNode(2, new ListNode(3)));
```
More Examples

ListNode Q, L;

L = new ListNode(3, null);
Q = L;

Q = new ListNode(42, null);
L.next = Q;

L.next.item += 1;
Insertion

- Inserting an item into the middle of a linked list takes *constant* amount of time
  - as long as you have a reference to the previous node
- List can grow until memory runs out.

```java
public void insertAfter(int item) {
    next = new ListNode(item, next);
}
```

l2.insertAfter(5);
Lookup

• Finding an item at position $n$ (random lookup) takes time proportional to $n$.
  
  – Remember this operation is constant time for an array.

• We’ll encounter data structures that finds a balance between arrays and linked lists.
public ListNode get(int position) {
    if (position < 0)
        return null;
    ListNode toReturn = this;
    for (int i = 0; i < position && toReturn != null; i++) {
        toReturn = toReturn.next;
    }
    return toReturn;
}

public ListNode get(int position) {
    if (position == 0) {
        return this;
    } else if ((position < 0) || (next == null)) {
        return null; // error checking
    } else {
        return next.get(position - 1);
    }
}
List Class

• **Why not just use ListNodes?**

• **The List class maintains:**
  – reference to the *head* of the List
  – *methods* to operate on the List
  – *information* about the List

```java
public class List {
    private ListNode head;
    private int size;

    public List() {
        head = null;
        size = 0;
    }

    public void insertFront(Object item) {
        head = new ListNode(item, head); size++;
    }
}
```
public class List {
    private ListNode head;
    private ListNode tail;
    private int size;

    public List() {
        head = null;
        tail = null;
        size = 0;
    }

    public void insertTail(int item) {
        if (head == null) // tail must be null too
            head = tail = new ListNode(item, null);
        else {
            tail.next = new ListNode(item, null);
            tail = tail.next;
        }
    }
}
Deletion

• **Deletion from the front of a list**

```java
public class List {
    public void deleteFront() {
        if (head != null) {
            head = head.next;
            size--;
        }
    }
}
```

• **Deletion from the end of the list?**
  – Must search through the entire list!
Doubly Linked List

For a doubly linked list each node has:

- an item
- a reference to the next node
- a reference to the previous node

```
public class DListNode {
    int item;
    DListNode next;
    DListNode prev;
}
```
DList Class

```java
class DList {
    private DListNode head;
    private DListNode tail;
}
```

```
tail.prev.next = null;
tail = tail.prev;
or
tail = tail.prev;
tail.next = null
```

This works only when there is at least 2 elements in the list. You need special cases for when the number of elements is 0 and 1.
Next Class

• Reading:
  – Objects, Abstraction, Data Structures Chapter 4

• Next Class:
  – More Lists