CS 3137, Class Notes

1 Linked Lists

- Linked lists represent an ordered set of elements. We can refer to the list as having a *head* element and *tail* element.
- The list ADT (Abstract Data Type) has a protocol that says you must access each member of the list to get to the next member.
- Common list processing primitive functions include *Insert (add), Remove (delete), Find, Count, IsEmpty, MakeEmpty, PrintList* etc.
- We can implement a list using an array, but it is not a good method. Each time we insert at as particular position k we need to shift the entire array to make room. Similarly, if we delete an element in the list, we need to shift all the elements to fill up the empty slot. We also have to allocate an array of a particualr size, even though we don't know how many elements we will need at any one time.
- Linked lists are a dynamic data structure, that can be created at run time without knowing ahead of time how many items are in the list. This is one of the most important and powerful uses of linked lists versus arrays: you don't need to know ahead of time how many elements to allocate as in an array.

- Most list operations are $O(N)$, where N is the number of list elements. For example, to find an element in an unordered list, you must potentially look at all N list elements, and on average, $N/2$ elements, which is $O(N)$.
- However, there is a cost in using linked lists. A small cost in extra memory needed to store the links to the next element in the list, and the cost of extra operations and complexity in traversing the lists.
- Boundary conditions are a very important part of list processing. Whenever we do list processing operations, we need to check for the special cases of first or last element in the list. For example, if we try to delete the first element of a single element list, the link to the head of the list will have to be updated. Similarly, if we need to insert at the front of the list, this also changes the link that points to the head of the list.
- An easy solution (some would call it a hack!) to many boundary condition problems is to always have an empty *header* node at the beginning of every list. Then, you will never have a pointer to a list that is NULL, since even empty lists have a single, empty, header node.
- Note that there is a native implementation of linked lists in the **java.util** package which implements a class **LinkedList**. We will be writing our own implementation of this package.

Here is a linked list routine that does its own linked list administration - DOES NOT USE Java's LinkedList classes:

```
//*******************************************************************
//  MagazineRack.java       Author: Lewis/Loftus
//
//  Driver to exercise the MagazineList collection.
//******************************************************************

public class MagazineRack
{
   //----------------------------------------------------------------
   //  Creates a MagazineList object, adds several magazines to the
   //  list, then prints it.
   //----------------------------------------------------------------
   public static void main (String[] args)
   {
      MagazineList rack = new MagazineList();

       rack.add (new Magazine("Time"));
       rack.add (new Magazine("Woodworking Today"));
       rack.add (new Magazine("Communications of the ACM"));
       rack.add (new Magazine("House and Garden"));
       rack.add (new Magazine("GQ"));

       System.out.println ("\n" +rack);

       rack.delete(new Magazine("Time"));
       rack.delete(new Magazine("GQ"));
       rack.delete(new Magazine("Communications of the ACM"));
       rack.add(new Magazine("People"));
       System.out.println ("\n" + rack);

       System.out.println ("Forming a new Magazine Rack, alphabetized:");

      MagazineList alpharack = new MagazineList();

      alpharack.addalpha(new Magazine("Fortune"));
      alpharack.addalpha(new Magazine("Newsweek"));
      alpharack.addalpha(new Magazine("Blender"));
      alpharack.addalpha(new Magazine("Jump"));
      alpharack.addalpha(new Magazine("Zagat's"));
      alpharack.addalpha(new Magazine("Zzagat's"));
      alpharack.addalpha(new Magazine("Zaagat's"));
      alpharack.addalpha(new Magazine("Zzzzgat's"));
      System.out.println ("\n" + alpharack);

      System.out.println("Magazine Rack Contents - print reversed:");
      alpharack.printListReverse();
      System.out.println();

      System.out.println ("Erasing Magazine Rack contents");
      alpharack.erase();
      System.out.println (alpharack);
   }
}
```

```
//*******************************************************************
//  MagazineList.java       Author: Lewis/Loftus
//
//  Represents a collection of magazines.
//******************************************************************

public class MagazineList
{
   private MagazineNode list;

   //----------------------------------------------------------------
   //  Sets up an initially empty list of magazines.
   //----------------------------------------------------------------
   MagazineList()
   {
      list = null;
   }

   //----------------------------------------------------------------
   //  Creates a new MagazineNode object and adds it to the end of
   //  the linked list.
   //----------------------------------------------------------------
   public void add (Magazine mag)
   {
      MagazineNode node = new MagazineNode (mag);
      MagazineNode current;
      System.out.println("adding at end " + mag);
      if (list == null)
         list = node;
      else
      {
         current = list;
         while (current.next != null)
            current = current.next;
         current.next = node;
      }
   }
   public void addalpha (Magazine mag)
   {
      MagazineNode node = new MagazineNode (mag); //get new node
      MagazineNode current,prev;

      System.out.println("adding alphabetical " + mag);
      if (list == null) //empty list, add at head of list
         list = node;
      else
      {
         prev=null;
         current = list;
         while (current!= null && current.magazine.compareTo(mag)<0){
            prev=current;
            current = current.next;
         }
         if(prev==null) {//goes in first if prev==null
             list=node; //list is first node in list
             node.next=current; // update new node to point to rest of list
         }
         else { // goes in after prev pointer
             prev.next=node;  //add after prev
             node.next=current; // update new node to point to rest of list
         }
      }
   }
```

```
     public void delete(Magazine mag)
     {
         MagazineNode current=list;
         MagazineNode prev;
         System.out.println("deleting " + mag);
         if(current==null) { // delete from empty list
            System.out.println("illegal delete from empty list");
         }
         else  if(current.magazine.equals(mag)){ //first entry deleted
                  list=current.next;
               } else {
                  prev=null;
                  while (current!=null &&  //loop down list looking for mag
                              !current.magazine.equals(mag)){
                     prev=current;
                     current=current.next;
                  }
                  if(current==null) { //end of list reached
                     System.out.println("magazine " + mag + "  not in list");
                  } else {
                     prev.next=current.next; //bypass deleted element
            }
        }
     }

    //-----------------------------------------------------------------
    //  Returns this list of magazines as a string.
    //-----------------------------------------------------------------
    public String toString ()
    {
       String result = "Magazine Rack contents:\n";
       MagazineNode current = list;

       while (current != null)
       {
          result += current.magazine + ", ";
          current = current.next;
       }
       result=result + "\n";
       return result;
    }

    public void erase(){ list=null;}


    public void printListReverse()
      {
          MagazineNode current=list;
          this.printReverse(current);
          System.out.println();
      }
    public void printReverse(MagazineNode current)
      {
          if(current!=null) {
            printReverse(current.next);
            System.out.print(current.magazine + ", ");
            }

       }
}
```

```
   //******************************************************************
   //  An inner class that represents a node in the magazine list.
   //  The public variables are accessed by the MagazineList class.
   //******************************************************************
    class MagazineNode
   {
      public Magazine magazine;
      public MagazineNode next;

      //--------------------------------------------------------------
      //  Sets up the node
      //--------------------------------------------------------------
      public MagazineNode (Magazine mag)
      {
         magazine = mag;
         next = null;
      }
   }

//*******************************************************************
//  Magazine.java       Author: Lewis/Loftus
//
//  Represents a single magazine.
//*******************************************************************
import java.lang.String;

public class Magazine implements Comparable
{
   private String title;
   //-----------------------------------------------------------------
   //  Sets up the new magazine with its title.
   //-----------------------------------------------------------------
   public Magazine (String newTitle)
   {
      title = newTitle;
   }

   //-----------------------------------------------------------------
   //  Returns this magazine as a string.
   //-----------------------------------------------------------------

   public boolean equals(Object Obj)
   {
       Magazine testMag = (Magazine) Obj;
//       System.out.println("testing " + testMag.title + " " + title);
       if (testMag.title.equals(title))
           return(true); else return false;
   }
   public String toString ()
   {
      return title;
   }

   public int compareTo(Object Obj)
   {
      Magazine testMag = (Magazine) Obj;
      return (title.compareTo(testMag.title));
   }
}
```

Here is the output of the program:

```
$java MagazineRack

adding at end Time
adding at end Woodworking Today
adding at end Communications of the ACM
adding at end House and Garden
adding at end GQ

Magazine Rack contents:
Time, Woodworking Today, Communications of the ACM, House and Garden, GQ,

deleting Time
deleting GQ
deleting Communications of the ACM
adding at end People

Magazine Rack contents:
Woodworking Today, House and Garden, People,

Forming a new Magazine Rack, alphabetized:
adding alphabetical Fortune
adding alphabetical Newsweek
adding alphabetical Blender
adding alphabetical Jump
adding alphabetical Zagat's
adding alphabetical Zzagat's
adding alphabetical Zaagat's
adding alphabetical Zzzzgat's

Magazine Rack contents:
Blender, Fortune, Jump, Newsweek, Zaagat's, Zagat's, Zzagat's, Zzzzgat's,

Magazine Rack Contents - print reversed:
Zzzzgat's, Zzagat's, Zagat's, Zaagat's, Newsweek, Jump, Fortune, Blender,

Erasing Magazine Rack contents
Magazine Rack contents:
```

2 Using Java Collections LinkedList class

Java collections has a class LinkedList that supports most LinkedList operations. As of Java 1.5, it can be generically typed as well.

This class gives you access to the first and last element of the list with **addFirst(Object o), addLast(Object o)** and **Object getFirst(), Object getLast()** methods.

To add and remove elements *anywhere* in the list, there is an iterator class that allows you to position yourself anywhere in the list. It does not give you direct access to the links themselves to make sure you don't mess them up and break the list apart.

```
LinkedList list = new LinkedList();
ListIterator iterator = list.listIterator();
```

You should think of the iterator as pointing **between** 2 adjacent items of the list. Initially, the iterator points just **before** the first item in the list. To move the iterator, use the method **iterator.next()**. To check to see if there is a next element, use the boolean method **iterator.hasNext()**

The **next()** method returns the object of the link that is is passing. The iterator **add(Object o)** method adds an object after the iterator, and then moves the iterator position past the new element. There is also a **remove()** method, but it is tricky. *Remove* removes and returns the object in the list that was returned by the last call to *next*. It can only be called once after a **next()**, and it cannot be called directly after an add operation.

Finally, this class is a doubly-linked list, with **previous()** as well as **next()** methods, meaning you can move in both directions in the list.

```
//  This program demonstrates operations on linked lists using Java LinkedList class

import java.util.*;
public class LinkedListTest2{
   public static void main(String[] args)   {
      LinkedList a = new LinkedList();
      a.add("Angela");
      System.out.println("adding to list a " + a);
      a.add("Carl");
      System.out.println("adding to list a " + a);
      a.add("Erica");
      System.out.println("adding to list a " + a);

      LinkedList b = new LinkedList();
      b.add("Bob");
      b.add("Doug");
      b.add("Frances");
      b.add("Gloria");
      System.out.println("list b: " + b);

// merge (interleave) items from lists a and b into list a
      System.out.println("merging list b into list a");
      ListIterator aIter = a.listIterator();
      ListIterator bIter = b.listIterator();
      while (bIter.hasNext())
      {
          if (aIter.hasNext()) aIter.next();
          aIter.add(bIter.next());
          System.out.println("mergedlist: " + a);
      }
      System.out.println("List a after merges: " + a);

// remove every second word from b
      bIter = b.listIterator();
      while (bIter.hasNext())   {
         bIter.next(); // skip one element
         if (bIter.hasNext())  {
            bIter.next(); // skip next element
            bIter.remove(); // remove that element
         }
      }
      System.out.println("List b with every 2nd word removed " + b);

// bulk operation: remove all words in b from a
      a.removeAll(b);
      System.out.println("remove all words in list b from list a: " + a);
   }
}
adding to list a [Angela]
adding to list a [Angela, Carl]
adding to list a [Angela, Carl, Erica]
list b: [Bob, Doug, Frances, Gloria]
merging list b into list a
mergedlist: [Angela, Bob, Carl, Erica]
mergedlist: [Angela, Bob, Carl, Doug, Erica]
mergedlist: [Angela, Bob, Carl, Doug, Erica, Frances]
mergedlist: [Angela, Bob, Carl, Doug, Erica, Frances, Gloria]
List a after merges: [Angela, Bob, Carl, Doug, Erica, Frances, Gloria]
List b with every 2nd word removed [Bob, Frances]
remove all words in list b from list a: [Angela, Carl, Doug, Erica, Gloria]
```