ECE 242
Data Structures and Algorithms

http://www.ecs.umass.edu/~polizzi/Teaching/ECE242/

Linked List IV
Lecture 13

Prof. Eric Polizzi
The big picture... so far

DATA STRUCTURES

- Array
  - 1d, 2d, etc.

- Linked-List: Simple, Doubly (today), etc.

ALGORITHMS

- Unsorted list
  - insert, delete, linear search, shuffle, sorting

- Sorted list
  - insert, delete, shuffle, binary search

- Stacks
  - push, pop, peek, etc.

- Queues
  - enqueue, dequeue, peek, etc.

- Priority Q
  - insert, delete, shuffle, binary search

Abstract Data Type (ADT) must implement User Interfaces
Simple Linked-List: Summary

- **Simple Linked-list**
  - insertFirst, O(1)
  - deleteFirst, O(1)
  - displayList, O(N)
  - find, O(N)
  - delete, O(N)

- **Simple Linked-list: double-ended feature**
  - insertLast, O(1)

- **Limitations:**
  - deleteLast is O(N)
  - No convenient way to traverse backward (go to previous link)...problematic for some practical applications; for example:
    - **Text Editor** (each line stored a string object embedded in a Link). Cursor can move down (next) but a simple Linked-list will not allow to have the cursor moving up easily.
Doubly Linked List

- Just a precision to avoid confusion:
  - (Doubly linked list) != (double-ended linked list)
- Allows to traverse the list forward and backward
- There are now two references within each Link instead of one
  - One reference to the next Link … next
  - One reference to the previous Link...prev
Doubly Linked List

- Doubly Linked List (using double-ended feature)

```java
class Link {
    public String name; // data item
    public Link next;  // reference to next link
    public Link prev;  // reference to previous link

    // Constructor
    public Link(String name){
        this.name = name;
        next = null;  // optional
        prev = null;  // optional
    }
}
```
Doubly Linked-List: Traversal

class DoublyLinkList {

    private Link first; // Reference to the first link
    private Link last;  // Reference to the last link

    // constructor
    public DoublyLinkList() { first = null; last = null; }

    // methods
    public void displayListForward() {
        Link current = first; // start probe at the beginning
        while (current != null) { // until the end of the list
            System.out.println(current.name);
            current = current.next; // move to next Link
        }
    }

    public void displayListBackward() {
        Link current = last; // start probe at the end
        while (current != null) { // until the end of the list
            System.out.println(current.name);
            current = current.prev; // move to next Link
        }
    }

    ...
}
public void insertFirst(String name) {
    Link newLink = new Link(name); // create link
    if (isEmpty())
        last = newLink; // special case
    else
        first.prev = newLink; // (step 1)
    newLink.next = first; // (step 2)
    first = newLink; // (step 3)
}
Doubly Linked-List: insertLast

Same process than insertFirst but applied at the end of the list (mirror image of insertFirst)

```java
public void insertLast(String name)
{
    Link newLink = new Link(name); // create link
    if (isEmpty())
        first= newLink; // special case
    else
        last.next = newLink; // (step 1)
    newLink.prev = last; // (step 2)
    last = newLink; // (step 3)
}
```
// Once current link is found do:

if (current == last)
    last = newLink; // special case
else
{
    newLink.next = current.next; // (step 1)
    current.next.prev = newLink; // (step 2)
}
newLink.prev = current;       // (step 3)
current.next = newLink;      // (step 4)
}
Doubly Linked-List: delete

// Once current link is found do:

if (current==first)
    first = current.next;
else
    current.prev.next = current.next;  // (step 1)

if (current==last)
    last = current.prev;
else
    current.next.prev = current.prev;  // (step 2)

- To do: deleteFirst, deleteLast
a- We consider the following doubly and double-ended linked list where we also provide the methods: displayForward, and insertFirst. Complete the deleteFirst and deleteLast methods.

```java
class Link {
    public int data; // data item
    public Link next; // reference to next link
    public Link prev; // reference to previous link

    // Constructor
    public Link(int data) {
        this.data = data;
        next = null; // optional
        prev = null; // optional
    }
}

class Doubly2EndedLinkList {
    private Link first; // Reference to the first link
    private Link last; // Reference to the last link

    // constructor
    public Doubly2EndedLinkList() { first = null; last = null; }

    // methods
    public void displayListForward() {
        Link current = first; // start probe at the beginning
        while (current != null) { // until the end of the list
            System.out.println(current.data);
            current = current.next; // move to next Link
        }
    }

    public void insertFirst(int data) {
        Link newLink = new Link(data); // create link
        if (isEmpty())
            last = newLink; // special case
        else
            first.prev = newLink; // {step 1}
        newLink.next = first; // {step 2}
        first = newLink; // {step 3}
    }

    public Link deleteFirst() {
        // implementation
    }

    public Link deleteLast() {
        // implementation
    }
}
```
Doubly Linked-List: Summary

- **Benefits:**
  - Traverse backward
  - More flexibility in removing, inserting objects
  - Deletion from the end of the list O(1)

- **Drawback**
  - Need to manage multiple pointers

- **Other implementation/configuration of linked-list possible**
  - Linklist of linklist
  - Circular Linked-list
  - Skiplist
  - Etc.
Iterators

- Direct manipulation of lists not desirable (prone to errors by the class user)
- We still want to offer the option to access a reference that can point to any arbitrary Link
- Iterator provides a clean abstraction for the user to operate the list

User can access a list iterator class to traverse the list and perform some operations such as
- getCurrent
- reset
- nextLink
- deleteCurrent, insertAfter, insertBefore
Iterators

```java
public static void main(String[] args) {
    LinkList mylist = new LinkList();
    ListIterator iter = mylist.getIterator();

    Link aLink = iter.getCurrent(); // access link at iterator
    iter.nextLink(); // move to next link
    iter.reset(); // set to start of the list
    ...
}
```

```java
class LinkList {
    ...
    public ListIterator getIterator() {
        return new ListIterator(this)
    }
    ...
}
```

```java
class ListIterator {
    private ListIterator() {
        ourList = list;
        current = ourList.getFirst();
        previous = null;
    }
    ...
}
```
Logistics

- Tuesday = Monday
- Class on Tuesday