Linked Lists and Iterators

2-22-2005

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment? Remember that the design is due today.
- Why would you want to use a singly linked list or any other type for that matter? Sorting linked lists?
- Linked data structures in general.

Code for the Singly Linked List

- Let's review the code for a basic singly linked list. In particular, note the recurring pattern of walking a list. That is a very important pattern for you to recognize though you won't be specifically using it much in the project.

Circular Linked Lists

- It is also possible to build a list where the "tail" points back to the "head". In this case those two terms really aren't all that well defined.
- Instead we can have a pointer anywhere in the list. We still can't walk backwards, but we can walk all the way around to get to anything we want.

Doubly Linked Lists

- Another variation on lists that can be useful is the doubly linked list. In a doubly linked list, every element knows both the one before it and the one after it. With this added in, you can delete an element without walking the list, or add one without having to go looking for the previous one.
- These require a bit more work.

Sentinels

- One way to help simplify linked list code, especially for doubly linked lists, is to add a sentinel.
- This is a special node that signifies an "end" of the list. We can put it at the beginning and the end by making it a circular list.
- Doubly linked lists with sentinels are perhaps that easiest type of list because they lack special case code

•Code for a Doubly Linked List

- Now let's look at code for a doubly linked list with a sentinel. You should note that this code is much simpler in many ways because it lacks all the special cases.

Iterators

- You have now seen patterns for walking through the elements of an array and a linked list. These are very significant patterns when we are dealing with low level code. However, they are also very different and can't be easily interchanged. We would like a pattern for walking through the elements of any container, whether it be an array, a linked list, or other things we will discuss later.
- To do this, we introduce the concept of an iterator.

Iterators Continued

- As the name implies, and iterator lets us iterate through the elements of a container. Java has an Interface called Iterator that has three methods. Let's go to the Java API to look at those methods.
- Java also has a similar construct with a less common name called an Enumeration. This was used in older Java libraries.

Using an Iterator

- If we have some type of container, **`cont`**, that can give us an Iterator then we can use the following loop structure to walk through the elements of that container, regardless of the nature of the container.
- for(Iterator iter=cont.iterator(); iter.hasNext();)
- What would an Iterator for an array based list look like? How about a linked list?

Sorting Linked Lists and Sorted Lists

- Like arrays, linked lists can be sorted. However, what is easy with each is different. The easiest sorts with linked lists build new lists instead of swapping pieces. (Insertion and Selection are easy.)
- You can also build lists that are always sorted. These have a different interface and fall into a completely different set of data structures which are associative.

Iterator Code

- Let's now go and write an iterator for one of our linked list classes.

Minute Essay

- The Iterator for a list can be a private inner class. How would this work? How can you use it outside of the class if it is private?
- Remember to generate your design today and put it out on the web. The working code is due Thursday.
- The midterm is one week from Thursday.
- Read java.awt and javax.swing.