Administrative Details

• Midterm will be returned this week
• Lab 6 posted today
  • Implement a Postscript-based calculator
  • Page 247 in Bailey (10.5 Laboratory: A Stack-Based Language)
Before Spring Break

- Discussed stacks, queues, and deques
- Infix vs. postfix expressions
  - Dijkstra’s Shunting Yard algorithm

Review lectures 17&18 (Bailey Chapter 10) before Wednesday’s lecture to prep for Lab 6
Today’s Outline

• Iterators (Bailey Ch 8)
  • Treat lectures an advertisement for the book
  • Reading the text before or after lecture is up to you, but the book is an important resource
    • So far we’ve covered chapters 1-10
Pre-midterm Review: Common Structure Operations

• size()
• isEmpty()
• add()
• remove()
• clear()
• contains()

• What’s missing?
  • Common method for efficient data traversal
  • iterator()
Visiting Data from Structure

• Write a method (numOccurs) that counts the number of times a particular Object appears in a structure

```java
public int numOccurs(List data, Object o) {
    int count = 0;
    for (int i=0; i<data.size(); i++) {
        Object obj = data.get(i);
        if (obj.equals(o)) count++;
    }
    return count;
}
```

• How does this fare on the structures that we have studied so far?
Problems?

- `get(i)` not defined on Linear structures (e.g., stacks and queues)
- `get(i)` is “slow” on some structures
  - O(n) on SLL (and DLL)
  - So `numOccurs = O(n^2)`

- How to balance generality with efficiency?
  - We want to be data structure-specific for efficiency
  - We want a common interface for generality
Iterators

- **Iterators** provide us with a common way to efficiently cycle through elements of a data structure

- An Iterator:
  - Provides generic methods to traverse elements
  - Abstracts away details of how to access structure
  - Uses different implementations for each structure

- As usual, we use both an Iterator interface and an AbstractIterator class
  - What purpose does each serve?
Iterator Interface

• hasNext()  returns true if the iterator has more elements to visit

• next()     Moves the iterator along the traversal; returns the next value considered

AbstractIterator Class

• get()      returns the next value considered

• reset()    reset iterator to the beginning
General Iterator Usage

```java
Iterator<E> iter = data.iterator();
...
while (iter.hasNext()) {
    E item = iter.next();
    ...
}
```
public int numOccurs (List data, Object o) {
    int count = 0;
    for (int i=0; i<data.size(); i++) {
        Object obj = data.get(i);
        if (obj.equals(o)) count++;
    }
    return count;
}

public int numOccurs (List data, Object o) {
    int count = 0;
    Iterator iter = data.iterator();
    while (iter.hasNext()) {
        if(o.equals(iter.next()))
            count++;
    }
    return count;
}
Iterator Implementations

• All specific implementations in structure5 extend AbstractlIterator (which implements Iterator)
• We need to define the methods labeled “abstract” for each data structure (i.e., get(), next(), hasNext(), and reset())

• Methods are specialized for specific data structures
  • Example: SLL
SinglyLinkedListIterator
class SinglyLinkedListIterator<E> extends AbstractIterator<E> {
    protected SinglyLinkedListElement<E> head, current;

    public SinglyLinkedListIterator(SinglyLinkedListElement<E> head) {
        this.head = head;
        reset();
    }

    public void reset() {
        current = head;
    }

    public E next() {
        E value = current.value();
        current = current.next();
        return value;
    }

    public boolean hasNext() {
        return current != null;
    }

    public E get() {
        return current.value();
    }
}

In SinglyLinkedList.java:

public Iterator<E> iterator() {
    return new SinglyLinkedListIterator<E>(head);
}
VectorIterator
class VectorIterator<E> extends AbstractIterator<E> {
    protected Vector<E> theVector;
    protected int current;

    public VectorIterator(Vector<E> v) {
        theVector = v;
        reset();
    }

    public void reset() {
        current = 0;
    }

    public boolean hasNext() {
        return current < theVector.size();
    }

    public E get() {
        return theVector.get(current);
    }

    public E next() {
        return theVector.get(current++);
    }
}

In Vector.java:

public Iterator<E> iterator() {
    return new VectorIterator<E>(this);
}
General Rules for Iterators

1. Traverse data structure in consistent order
2. **Always call hasNext() before calling next()!!!**
3. Never change underlying data structure while iterating over it

- **Take away messages:**
  - Iterator objects capture state of traversal
  - They have access to internal data representations
  - Should be fast and easy to use
New Loop Syntax

Vector<String> words = new Vector<String>();
...
for(Iterator<String> i = words.iterator(); i.hasNext(); ) {
    String item = i.next();
    System.out.println(item);
}

Vector<String> words = new Vector<String>();
...
for (String word : words) {
    System.out.println(word);
}
More Iterator Examples

• How would we StackArrayIterator?
  • Do we go from bottom to top, or top to bottom?
  • Doesn’t matter! We just have to be consistent…

• We can also make “specialized iterators”
  • Filtering iterators