Linked Data Structures II: Doubly-Linked Lists
Sequential Access

• *Sequential access* usually means accessing the entries of a collection (with a *string* model) in increasing order of position, by accessing the “next” entry in the collection.

• Sometimes you can access the entries sequentially in the reverse direction, too, by accessing the “previous” entry in the collection.
  – Example: the OSU CSE components *List*
Interfaces and Classes

Standard

Iterable

ListKernel

List

List1L

List3
Interfaces and Classes

Standard

Iterable

Kernel

List

List1L

List3

Standard has contracts for three methods:
clear
newInstance
transferFrom
**ListKernel** has contracts for six methods:
- `addRightFront`
- `removeRightFront`
- `advance`
- `moveToStart`
- `leftLength`
- `rightLength`
List has contracts for five other methods:
- rightFront
- replaceRightFront
- moveToFinish
- retreat
- swapRights
LIST\_MODEL is (  
    left: \textit{string of} T,  
    right: \textit{string of} T  
)  

\textit{type} ListKernel \textit{is modeled by} LIST\_MODEL
Mathematical Model

LIST_MODEL is (  
   left: string of T,  
   right: string of T  
 )

You may think of these two strings as being to the left and right, respectively, of the “current position”.

type ListKernel is modeled by LIST_MODEL
No-argument Constructor

• Ensures:

\[ \textit{this} = (< >, < >) \]
void advance()

- Advances the position in this by one.
- Updates: this
- Requires:
  
  this.right /= < >

- Ensures:
  
  this.left * this.right = 
  
  #this.left * #this.right and
  
  |this.left| = |#this.left| + 1
**retreat**

```c
void retreat()
```

- Retreats the position in `this` by one.
- Updates: `this`
- Requires:
  ```
  this.left /= < >
  ```
- Ensures:
  ```
  this.left * this.right =
  #this.left * #this.right and
  |this.left| = |#this.left| - 1
  ```
What’s New?

• With just advance, sequential access is only to the “next” position
  – A singly-linked list representation provides good performance

• With retreat as well as advance, sequential access is also to the “previous” position
  – A singly-linked list representation provides poor performance
What’s New?

• With just \texttt{advance}, sequential access is only to the “next” position.
  – A singly-linked list representation provides good performance.

• With \texttt{retreat} as well as \texttt{advance}, sequential access is also to the “previous” position.
  – A singly-linked list representation provides poor performance.

To see why, write an implementation of \texttt{retreat} using only the \texttt{ListKernel} methods.
Example: **List2 (SLL)**

\[
\text{this} = (<18>, <6>)
\]
Example: \texttt{List2 (SLL)}

\texttt{this} = (<18>, <6>)

The abstraction function (correspondence) ...
Example: List2 (SLL)

data

? data

next

18 data

next

6 data

next

The “current position” is indicated by `this.lastLeft`.

\[ \text{this} = (<18>, <6>) \]
A Second Smart Node

- Note that the code for `Queue2` has no special cases at all, but the code for `List2` needs to handle a special case in `addRightFront` and `removeRightFront`.

- This can be eliminated by introducing a smart node at the end of the singly-linked list, too, so the two smart nodes are like “bookends”.
A Second Smart Node

- Note that the code for Queue2 has no special cases at all, but the code for List2 needs to handle a special case in addRightFront and removeRightFront.
- This can be eliminated by introducing a smart node at the end of the singly-linked list, too, so the two smart nodes are like “bookends.”

You should be able to re-write this code for List2 with two smart nodes, as illustrated on the next slide.
Example: SLL “Bookends”

\[
\text{this} = (\langle 18 \rangle, \langle 6 \rangle)
\]
Example: SLL “Bookends”

\[ \text{this} = (18, 6) \]

There is really no need for a \textbf{null} reference any more; the \? here means “unused”.\]
Doubly-Linked Lists

• In addition to the second smart node, the code for \texttt{List3} introduces one other (major) change

• The data structure is now a \textit{doubly-linked list}, in which there are two references per node: one to the “next” node and one to the “previous” node
  – This allows \texttt{retreat} to be implemented efficiently
Example: **List3 (DLL)**

\[ this = (\langle 18 \rangle, \langle 6 \rangle) \]
Resources

• Wikipedia: Linked Data Structure
  – http://en.wikipedia.org/wiki/Linked_data_structure

• *Big Java Late Objects*, Section 16.1 (but not the part about iterators)