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**
  - has contracts for three methods:
    - `clear`
    - `newInstance`
    - `transferFrom`

- **Iterable**

- **Kernel**
  - extends **Standard**

- **List**
  - extends **Kernel**

- **List1L**
  - implements **List**

- **List3**
  - implements **List**
ListKernel has contracts for six methods:
addRightFront removeRightFront advance moveToStart leftLength rightLength

List1L ... List3
List has contracts for five other methods:
- rightFront
- replaceRightFront
- moveToFinish
- retreat
- swapRights
Mathematical Model

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

type ListKernel 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:

\[
\text{this} = (\langle \rangle, \langle \rangle)
\]
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
void retreat()

• Retreats the position in this by one.
• Updates: this
• Requires:
  \[
  this . left /= < >
  \]
• Ensures:
  \[
  this . left * this . right =
  #this . left * #this . right \text{ and}
  |this . left| = |#this . left| - 1
  \]
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 `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

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

\[ \text{this} = (\langle 18 \rangle, \langle 6 \rangle) \]
Example: List2 (SLL)

this = (<18>, <6>)

The abstraction function (correspondence) ...
Example: \textbf{List2 (SLL)}

this $= (<18>, <6>)$

The “current position” is indicated by \texttt{this.lastLeft}.
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”

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

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

There is really no need for a null reference any more; ? here means “unused”.
Doubly-Linked Lists

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

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

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

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example.png}
\end{figure}
Resources

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

• Big Java (4th ed), Section 15.2 (but not the part about iterators)
  – https://library.ohio-state.edu/record=b8540788~S7