UNIVERSIDAD NACIONAL DE COLOMBIA

Estructuras de Datos

Sesión 3

List Data Structure (Part 3)

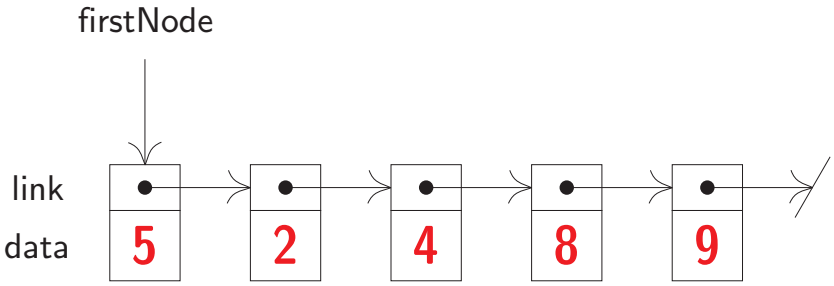
Yoan Pinzón

Table of Content Session 3

LinearList Data Structure

Linked Representation

- Each element of an instance of a data object is represented as a group of memory locations called **cell** or **node**.
- The elements may be stored in any arbitrary set of memory locations.
- Each element keeps explicit information about the location of the next element through a **link** (or **pointer**).
- This data structure is also called **chain**



Singly linked list or chain

Class Definition of ChainNode

```
5  package unal.datastructures;

7  class ChainNode<T>
8  {
9     // package visible fields
10    T element;
11    ChainNode<T> next;

13    // package visible constructors
14    ChainNode( )
15    {
16       this( null, null );
17    }

19    ChainNode( T element )
20    {
21       this( element, null );
22    }
```

```
24    ChainNode( T element, ChainNode<T> next )
25    {
26       this.element = element;
27       this.next = next;
28    }
29 }
```

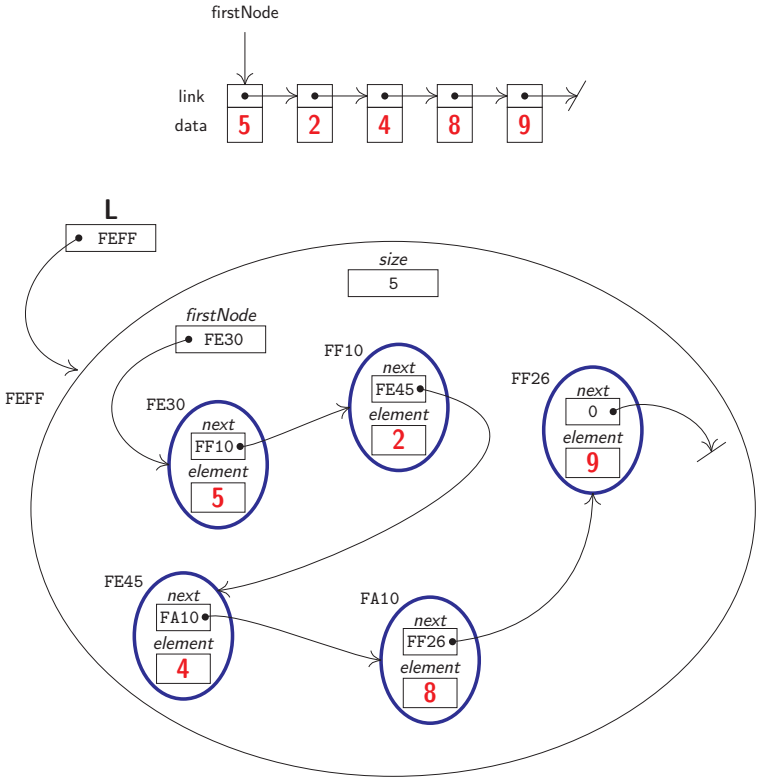

Class Definition of Chain

```
package unal.datastructures;

import java.util.*;

public class Chain<T> implements LinearList<T>, Iterable<T>
{
   // fields
   protected ChainNode<T> firstNode;
   protected int size;

   // constructor
   public Chain( ) { /* ... */ };

   // methods
   public boolean isEmpty( ) { /* ... */ }
   public int size( ) { /* ... */ }
```

```
    void checkIndex( int index ) { /* ... */ }
    public T get( int index ) { /* ... */ }
    public int indexOf( T theElement ) { /* ... */ }
    public T remove( int index ) { /* ... */ }
    public void add( int index, T theElement ) { /* ... */ }
    public String toString( ) { /* ... */ }
    public Iterator<T> iterator( ) { /* ... */ }
    private class ChainIterator implements Iterator<T> { /* ... */ }
    public static void main( String[] args ) { /* ... */ }
}
```

Note: This representation does not specify the maximum size!

An instance of this class (Chain L of integers) with `size=5` will look like this:



contructor

```
14    /** create a list that is empty */
15    public Chain( )
16    {
17       firstNode = null;
18       size = 0;
19    }
```


isEmpty

```
22    /** @return true iff list is empty */
23    public boolean isEmpty( )
24    {
25       return size == 0;
26    }
```

size

```
28    /** @return current number of elements in list */
29    public int size( )
30    {
31       return size;
32    }
```


checkIndex

```
34    /** @throws IndexOutOfBoundsException when
35      * index is not between 0 and size - 1 */
36    void checkIndex( int index )
37    {
38       if( index < 0 || index >= size )
39          throw new IndexOutOfBoundsException
40                ( "index = " + index + "  size = " + size );
41    }
```

get

```
43  /** @return element with specified index
44    * @throws IndexOutOfBoundsException when
45    * index is not between 0 and size - 1 */
46  public T get( int index )
47  {
48     checkIndex( index );

50     // move to desired node
51     ChainNode<T> currentNode = firstNode;
52     for( int i = 0; i < index; i++ )
53        currentNode = currentNode.next;

55     return currentNode.element;
56  }
```

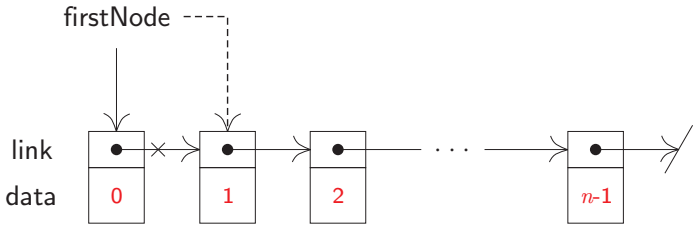

indexOf

```
58  /** @return index of first occurrence of theElement,
59    * return -1 if theElement not in list */
60  public int indexOf( T theElement )
61  {
62     // search the chain for theElement
63     ChainNode<T> currentNode = firstNode;
64     int index = 0; // index of currentNode
65     while( currentNode != null &&
66           !currentNode.element.equals( theElement ) )
67     {
68        // move to next node
69        currentNode = currentNode.next;
70        index++;
71     }
72     // make sure we found matching element
73     if( currentNode == null )
74        return -1;
75     else
76        return index;
77  }
```

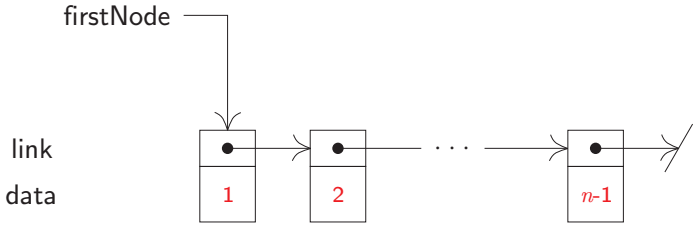
Deleteing the $index$th element

There are two cases to consider:

1) If $index$=0

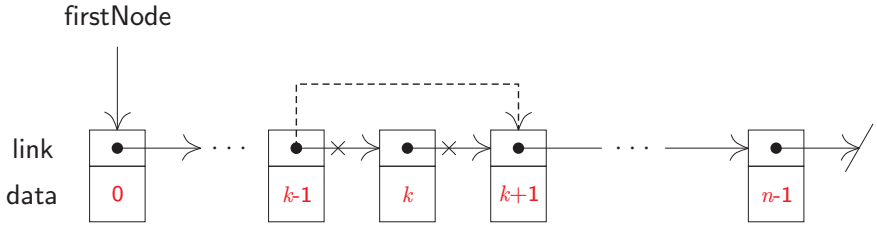


(a) Before

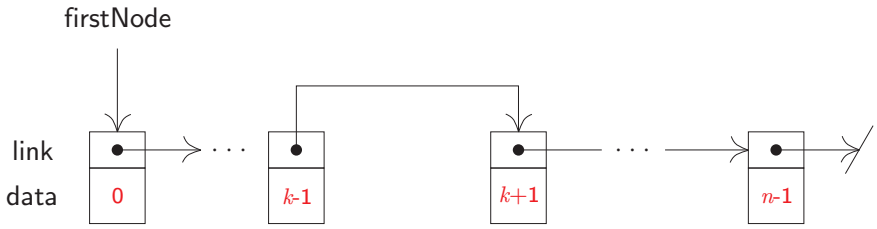


(b) After

2) If $index > 0$



(a) Before



(b) After

remove

```
79    /** Remove the element with specified index.
80      * All elements with higher index have their
81      * index reduced by 1.
82      * @throws IndexOutOfBoundsException when
83      * index is not between 0 and size - 1
84      * @return removed element */
85    public T remove( int index )
86    {
87       checkIndex( index );

89       T removedElement;
90       if( index == 0 ) // remove first node
91       {
92          removedElement = firstNode.element;
93          firstNode = firstNode.next;
94       }
95       else
96       {  // use q to get to predecessor of desired node
97          ChainNode<T> q = firstNode;
98          for( int i = 0; i < index - 1; i++ )
```

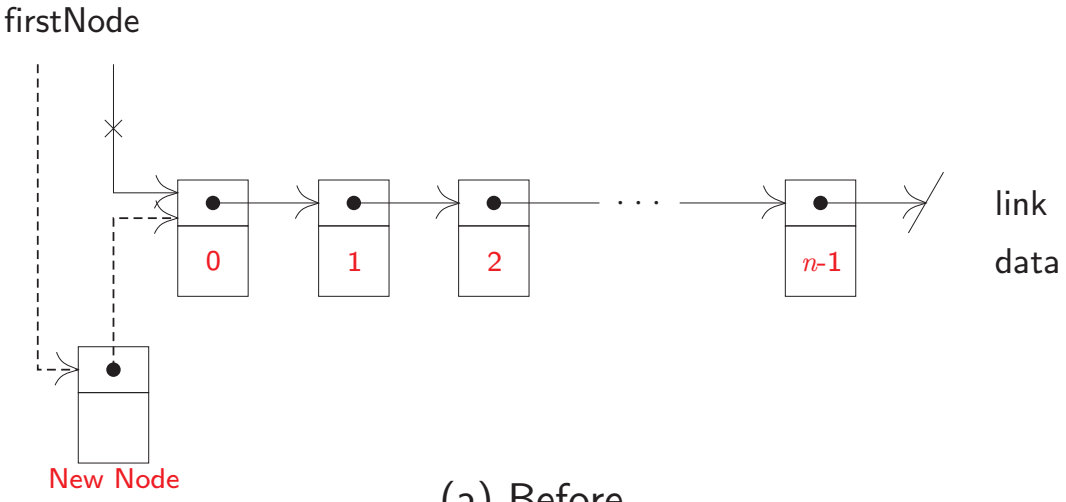


```
99             q = q.next;

101         removedElement = q.next.element;
102         q.next = q.next.next; // remove desired node
103      }
104      size--;
105      return removedElement;
106   }
```

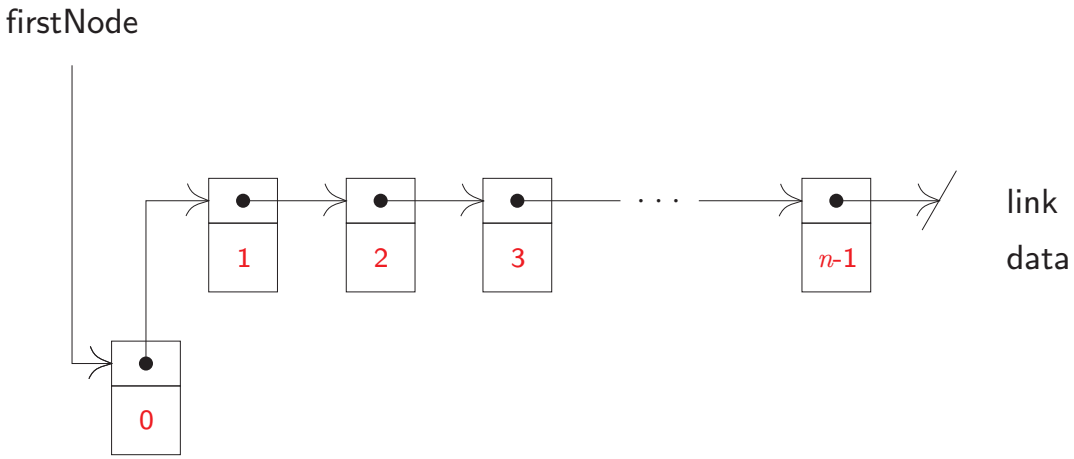
Inserting an Element

Insertion and removal work in a similar way. There are two cases to consider:

1) If $index$=0

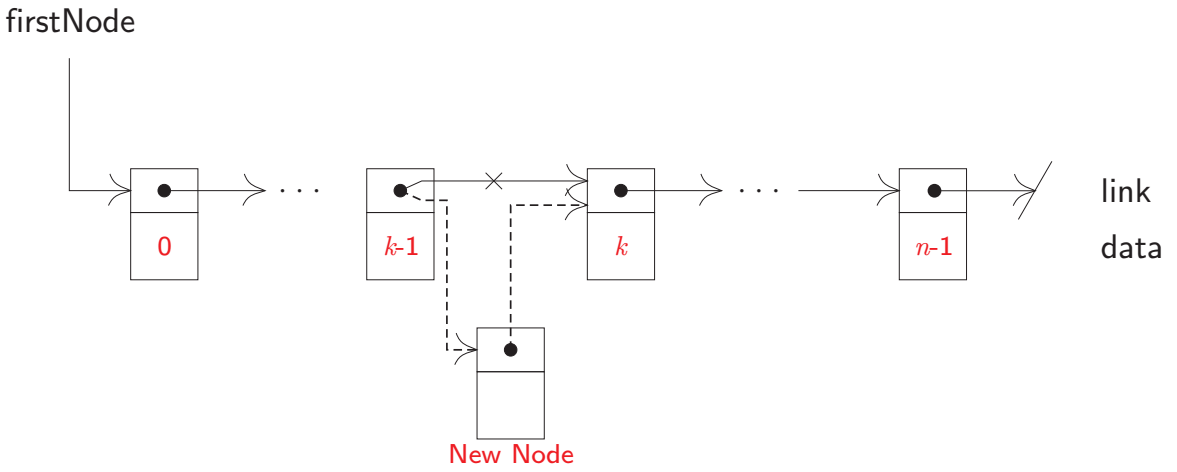


(a) Before

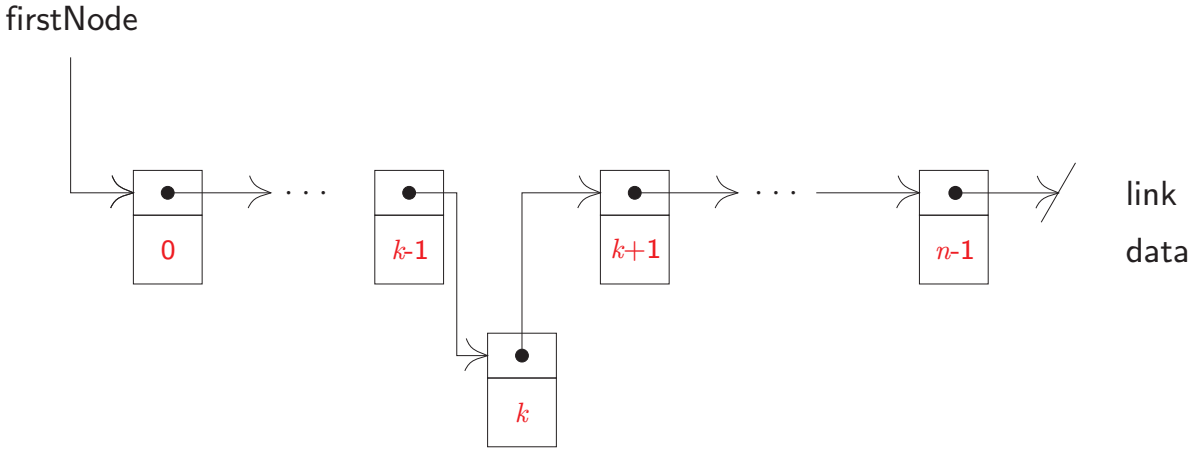


(b) After

2) If $index > 0$



(a) Before



(b) After

add

```
108   /** Insert an element with specified index.
109     * All elements with equal or higher index
110     * have their index increased by 1.
111     * @throws IndexOutOfBoundsException when
112     * index is not between 0 and size */
113   public void add( int index, T theElement )
114   {
115      if( index < 0 || index > size )
116         // invalid list position
117         throw new IndexOutOfBoundsException
118               ( "index = " + index + "  size = " + size );

120      if( index == 0 )
121         // insert at front
122         firstNode = new ChainNode<T>( theElement, firstNode );
123      else
124      {   // find predecessor of new element
125         ChainNode<T> p = firstNode;
126         for( int i = 0; i < index - 1; i++ )
```



```
127            p = p.next;

129         // insert after p
130         p.next = new ChainNode<T>( theElement, p.next );
131      }
132      size++;
133   }
```

toString

```
135   /** convert to a string */
136   @Override
137   public String toString( )
138   {
139      StringBuilder s = new StringBuilder( "[" );

141      // put elements into the buffer
142      for( T x : this )
143         s.append( Objects.toString( x ) + ",␣" );

145      if( size > 0 )
146         s.setLength( s.length( ) - 2 ); // remove last ", "

148      s.append( "]" );

150      // create equivalent String
151      return new String( s );
152   }
```


iterator

```
154   /** create and return an iterator */
155   public Iterator<T> iterator( )
156   {
157      return new ChainIterator( );
158   }
```

class ChainIterator

```
160   /** chain iterator */
161   private class ChainIterator implements Iterator<T>
162   {
163      // data member
164      private ChainNode<T> nextNode;

166      // constructor
167      public ChainIterator( )
168      {
169         nextNode = firstNode;
170      }

172      // methods
173      /** @return true iff list has a next element */
174      public boolean hasNext( )
175      {
176         return nextNode != null;
177      }

179      /** @return next element in list
```



```
180        * @throws NoSuchElementException
181        * when there is no next element */
182      public T next( )
183      {
184         if( nextNode != null )
185         {
186            T elementToReturn = nextNode.element;
187            nextNode = nextNode.next;
188            return elementToReturn;
189         }
190         else
191            throw new NoSuchElementException( "No next element" );
192      }

194      /** unsupported method */
195      public void remove( )
196      {
197         throw new UnsupportedOperationException
198               ( "remove not supported" );
199      }
200   }
```

main

```
202   /** test program */
203   public static void main( String[] args )
204   {
205      // test default constructor
206      Chain<Integer> x = new Chain<>( );

208      // test size
209      System.out.println( "Initial␣size␣is␣" + x.size( ) );

211      // test isEmpty
212      if( x.isEmpty( ) )
213         System.out.println( "The␣list␣is␣empty" );
214      else System.out.println( "The␣list␣is␣not␣empty" );

216      // test put
217      x.add( 0, new Integer( 2 ) );
218      x.add( 1, new Integer( 6 ) );
219      x.add( 0, new Integer( 1 ) );
220      x.add( 2, new Integer( 4 ) );
```



```
221      System.out.println( "List␣size␣is␣" + x.size( ) );

223      // test toString
224      System.out.println( "The␣list␣is␣" + x );

226      // test indexOf
227      int index = x.indexOf( new Integer( 4 ) );
228      if( index < 0 )
229         System.out.println( "4␣not␣found" );
230      else System.out.println( "The␣index␣of␣4␣is␣" + index );

232      index = x.indexOf( new Integer( 3 ) );
233      if( index < 0 )
234         System.out.println( "3␣not␣found" );
235      else System.out.println( "The␣index␣of␣3␣is␣" + index );

237      // test get
238      System.out.println( "Element␣at␣0␣is␣" + x.get( 0 ) );
239      System.out.println( "Element␣at␣3␣is␣" + x.get( 3 ) );

241      // test remove
242      System.out.println( x.remove( 1 ) + "␣removed" );
```

```
243       System.out.println( "The␣list␣is␣" + x );
244       System.out.println( x.remove( 2 ) + "␣removed" );
245       System.out.println( "The␣list␣is␣" + x );

247       if( x.isEmpty( ) )
248          System.out.println( "The␣list␣is␣empty" );
249       else System.out.println( "The␣list␣is␣not␣empty" );

251       System.out.println( "List␣size␣is␣" + x.size( ) );

253       // output using an iterator
254       Iterator y = x.iterator( );
255       System.out.print( "The␣list␣is␣" );
256       while( y.hasNext( ) )
257          System.out.print( y.next( ) + "␣" );
258       System.out.println( );
259    }
```


Compiling `Chain.java`

```
C:\2016699\code> javac unal\datastructures\Chain.java ↵
C:\2016699\code> java unal.datastructures.Chain ↵
Initial size is 0
The list is empty
List size is 4
The list is [1, 2, 4, 6]
The index of 4 is 2
3 not found
Element at 0 is 1
Element at 3 is 6
2 removed
The list is [1, 4, 6]
6 removed
The list is [1, 4]
The list is not empty
List size is 2
```