Lecture 15: Linked Lists and ADT List

CS2504/CS4092– Algorithms and Linear Data Structures

Dr Kieran T. Herley

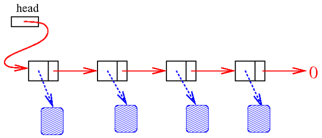
Department of Computer Science
University College Cork

2013/14

Summary

Doubly-linked list concept. Representation and implementation of ADT List using such lists.

Linked Lists and ADT List



Singly linked list; each node points to one list element

get Walk list until spec. element found, then return it

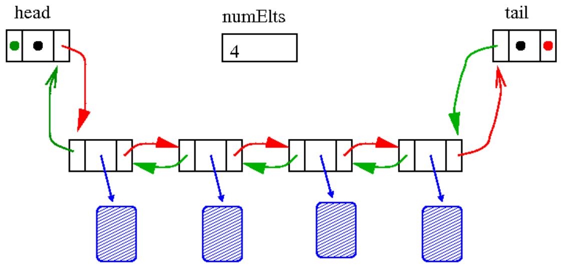
add Locate insertion point, splice in new node

remove Locate removal node, exscise it from list

Note: add/remove “surgery” must manipulate nodes “neighbouring” insertion/removal point– somewhat awkward with singly linked lists (try it!)

A Better Idea

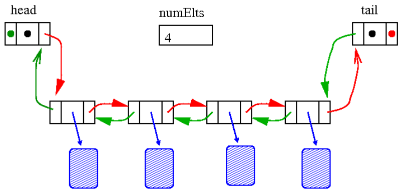
Use a *doubly-linked* list: nodes points both to predecessors and successors in the list.



Note: ● denotes the null pointer (diagramatic equiv. to $\rightarrow\emptyset$

It will also prove convenient to delimit the beginning and end of the list with special ("dummy") nodes head and tail. These nodes have no elements.

Doubly-Linked Lists and ADT List

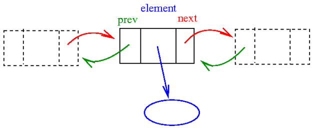


Note: ● denotes the null pointer (diagramatic equiv. to $\rightarrow \emptyset$

- doubly-linked list of nodes each containing a list element (object)
- nodes linked to predecessors (by prev ref) and successors (by next ref.)
- “dummy” nodes (with no element)
  - head– marks beginning of list
  - tail– marks end of list

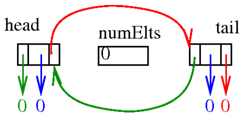
LLNode Revisited

```
public class LLNode<EltType>
{  public LLNode()
   {  element = null; prev = null; next = null;
   }
   public LLNode(LLNode<EltType> p,
                 LLNode<EltType> n,
                 EltType s)
   {  element = s;
      prev = p;
      next = n;
   }
   // public getters and setters
   private EltType element;
   private LLNode<EltType> prev;
   private LLNode<EltType> next;
}
```

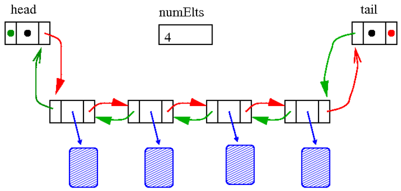


LinkedList

- An empty list:



- A non-empty list:



Note: ● denotes the null pointer (diagramatic equiv. to $\rightarrow\emptyset$

- Note: only head, tail and numElts are instance variables of LinkedList object.

ADT List

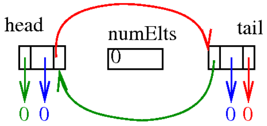
```
public interface List<EltType>
{  public int size();
   public boolean isEmpty();
   public EltType get(int inx);
   public EltType set(int inx, EltType newElt);
   public void add(EltType newElt);
   public void add(int inx, EltType newElt);
   public EltType remove(int inx);
   . . .
}
```

Simplified version of interface for ADT List

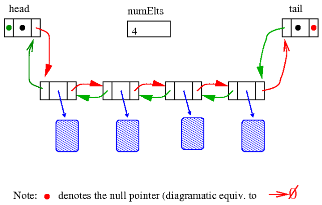
LinkedList Implementation

```
public class LinkedList<EltType>
        implements List<EltType>
{   public LinkedList ()
    {   head = new LLNode<EltType>(
                    null , null , null );
        tail  = new LLNode<EltType>(
                    head, null , null );
        head.setNext( tail ); numElts = 0;

    }
    . . .
    protected int numElts;
    protected LLNode<EltType> head;
    protected LLNode<EltType> tail;
}
```

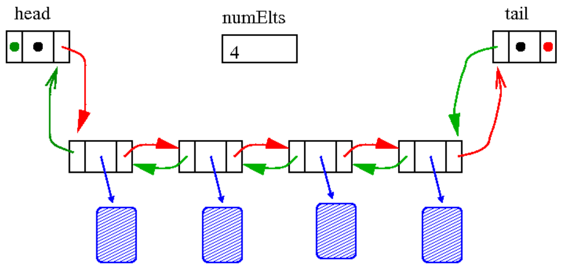


Method get



```
/* Return node with specified index within list */
private LLNode<EltType> nodeAtIndex(int index)
{ . . .
}
public EltType get(int inx)
{ LLNode<EltType> node = nodeAtIndex(inx);
  return node.element();
}
```

Method get cont'd



Note: ● denotes the null pointer (diagramatic equiv. to $\rightarrow\emptyset$

```
private LLNode<EltType> nodeAtIndex(int index)
{  LLNode<EltType> node = head.getNext();
   for (int i = 0; i < index; i++)
       node = node.getNext();
   return node;
}
```

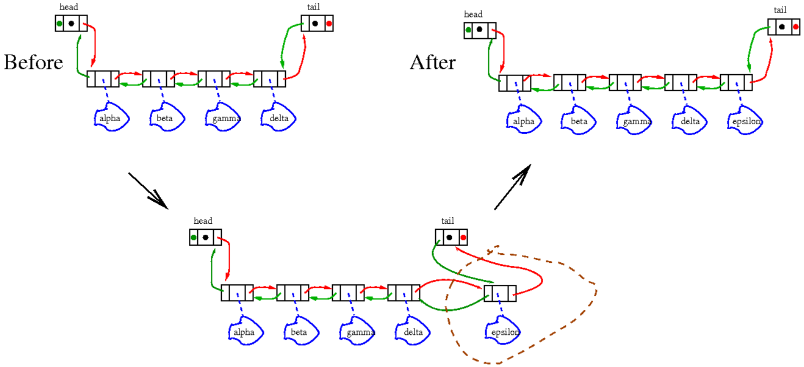
add Methods

add(newElt): Add element newElt at the end of the list. *Input: E*; *Output: None*.

add(inx, newElt): Add element newElt to the list at index inx. Illegal if inx is negative or greater than current list size[1]. *Input: int, E*; *Output: None*.

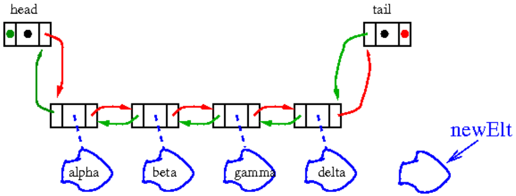
Adding at the End

Operation: add(“epsilon”)



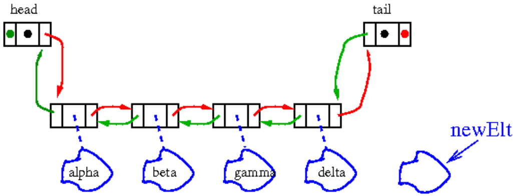
Adding at the End cont'd

```
public void add(EltType newElt)
{   numElts++;
    LLNode<EltType> oldLast = tail.getPrev();
    LLNode<EltType> newNode =
        new LLNode<EltType>(oldLast, tail, newElt);
    oldLast.setNext(newNode);
    tail.setPrev(newNode);
}
```



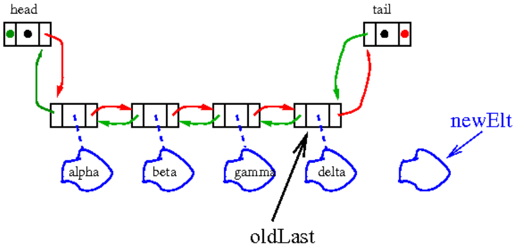
Adding at the End cont'd

```
public void add(EltType newElt)
{   numElts++;
    LLNode<EltType> oldLast = tail.getPrev();
    LLNode<EltType> newNode =
        new LLNode<EltType>(oldLast, tail, newElt);
    oldLast.setNext(newNode);
    tail.setPrev(newNode);
}
```



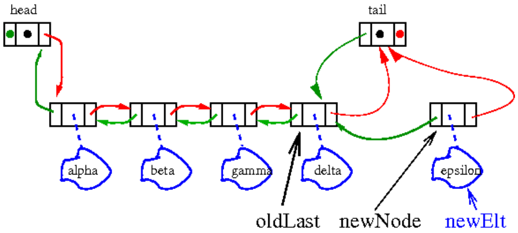
Adding at the End cont'd

```
public void add(EltType newElt)
{  numElts++;
   LLNode<EltType> oldLast = tail.getPrev();
   LLNode<EltType> newNode =
      new LLNode<EltType>(oldLast, tail, newElt);
   oldLast.setNext(newNode);
    tail.setPrev(newNode);
}
```



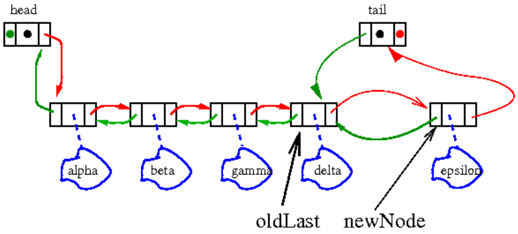
Adding at the End cont'd

```
public void add(EltType newElt)
{  numElts++;
   LLNode<EltType> oldLast = tail.getPrev();
   LLNode<EltType> newNode =
      new LLNode<EltType>(oldLast, tail, newElt);
   oldLast.setNext(newNode);
   tail.setPrev(newNode);
}
```



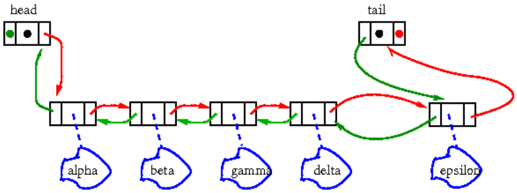
Adding at the End cont'd

```
public void add(EltType newElt)
{   numElts++;
    LLNode<EltType> oldLast = tail.getPrev();
    LLNode<EltType> newNode =
        new LLNode<EltType>(oldLast, tail, newElt);
    oldLast.setNext(newNode);
    tail.setPrev(newNode);
}
```



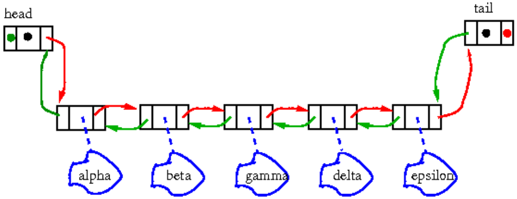
Adding at the End cont'd

```
public void add(EltType newElt)
{   numElts++;
    LLNode<EltType> oldLast = tail.getPrev();
    LLNode<EltType> newNode =
        new LLNode<EltType>(oldLast, tail, newElt);
    oldLast.setNext(newNode);
    tail.setPrev(newNode);
}
```



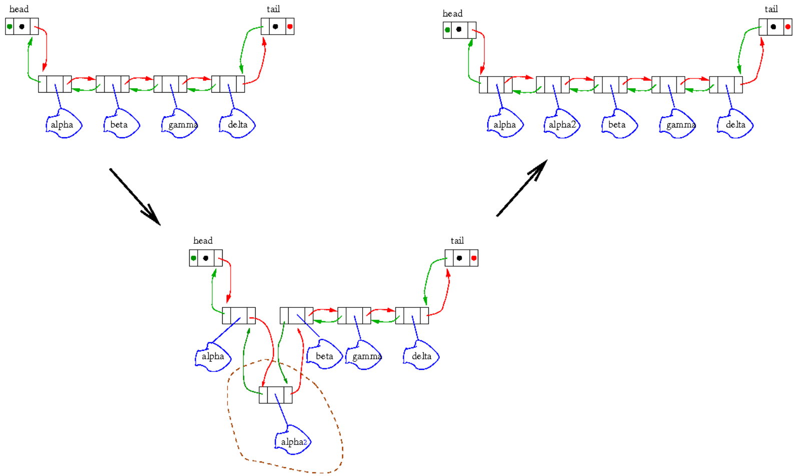
Adding at the End cont'd

```
public void add(EltType newElt)
{  numElts++;
   LLNode<EltType> oldLast = tail.getPrev();
   LLNode<EltType> newNode =
      new LLNode<EltType>(oldLast, tail, newElt);
   oldLast.setNext(newNode);
   tail.setPrev(newNode);
}
```



Adding Elsewhere

Operation add(1, "alpha2")

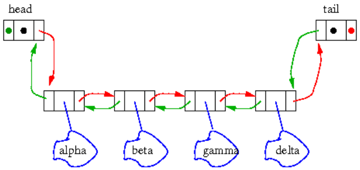


Method add

```
public void add(int inx, EltType newElt)
{   if (inx == size())
    {   add(newElt);
    }
    else
    {    LLNode<EltType> next = nodeAtIndex(inx);
         LLNode<EltType> prev = next.getPrev();
         LLNode<EltType> node =
              new LLNode<EltType>(prev, next, newElt);
         next.setPrev(node);
         prev.setNext(node);
         numElts++;
    }
}
```

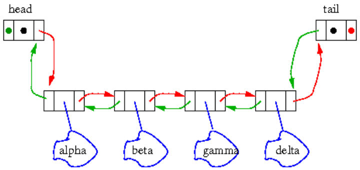
Method add cont'd

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {   LLNode<EltType> next = nodeAtIndex(inx);
        LLNode<EltType> prev = next.getPrev();
        LLNode<EltType> node =
            new LLNode<EltType>(prev, next, newElt);
        next.setPrev(node);
        prev.setNext(node);
        numElts++;
    }
}
```



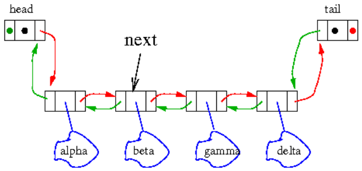
Method add cont'd

```
public void add(int inx, EltType newElt)
{  if (inx == size()){ . . . }
   else
   {   LLNode<EltType> next = nodeAtIndex(inx);
       LLNode<EltType> prev = next.getPrev();
       LLNode<EltType> node =
           new LLNode<EltType>(prev, next, newElt);
       next.setPrev(node);
       prev.setNext(node);
       numElts++;
   }
}
```



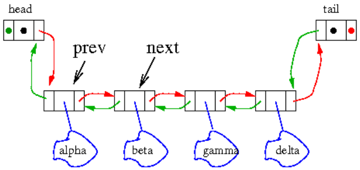
Method add cont'd

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {    LLNode<EltType> next = nodeAtIndex(inx);
         LLNode<EltType> prev = next.getPrev();
         LLNode<EltType> node =
              new LLNode<EltType>(prev, next, newElt);
         next.setPrev(node);
         prev.setNext(node);
         numElts++;
    }
}
```



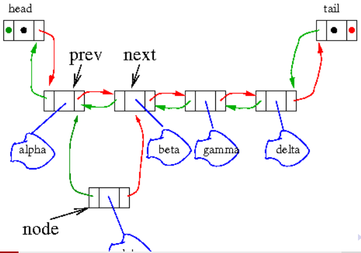
Method add cont’d

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {   LLNode<EltType> next = nodeAtIndex(inx);
        LLNode<EltType> prev = next.getPrev();
        LLNode<EltType> node =
            new LLNode<EltType>(prev, next, newElt);
        next.setPrev(node);
        prev.setNext(node);
        numElts++;
    }
}
```



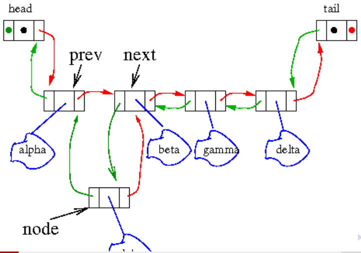
Method add cont'd

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {   LLNode<EltType> next = nodeAtIndex(inx);
        LLNode<EltType> prev = next.getPrev();
        LLNode<EltType> node =
            new LLNode<EltType>(prev, next, newElt);
        next.setPrev(node);
        prev.setNext(node);
        numElts++;
    }
}
```



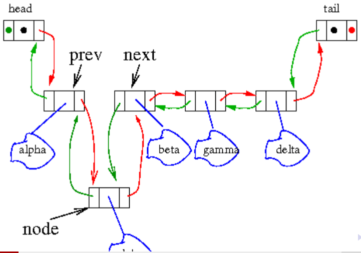
Method add cont'd

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {   LLNode<EltType> next = nodeAtIndex(inx);
        LLNode<EltType> prev = next.getPrev();
        LLNode<EltType> node =
            new LLNode<EltType>(prev, next, newElt);
        next.setPrev(node);
        prev.setNext(node);
        numElts++;
    }
}
```



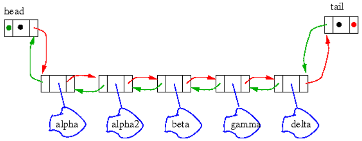
Method add cont'd

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {   LLNode<EltType> next = nodeAtIndex(inx);
        LLNode<EltType> prev = next.getPrev();
        LLNode<EltType> node =
            new LLNode<EltType>(prev, next, newElt);
        next.setPrev(node);
        prev.setNext(node);
        numElts++;
    }
}
```



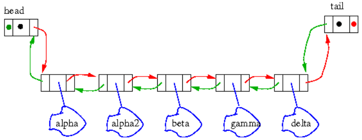
Method add cont'd

```
public void add(int inx, EltType newElt)
{   if (inx == size()){ . . . }
    else
    {   LLNode<EltType> next = nodeAtIndex(inx);
        LLNode<EltType> prev = next.getPrev();
        LLNode<EltType> node =
            new LLNode<EltType>(prev, next, newElt);
        next.setPrev(node);
        prev.setNext(node);
        numElts++;
    }
}
```



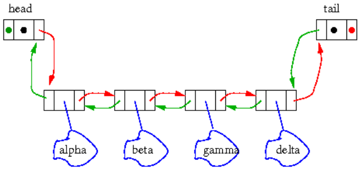
Method add cont'd

```
public void add(int inx, EltType newElt)
{  if (inx == size()){ . . . }
   else
   {   LLNode<EltType> next = nodeAtIndex(inx);
       LLNode<EltType> prev = next.getPrev();
       LLNode<EltType> node =
           new LLNode<EltType>(prev, next, newElt);
       next.setPrev(node);
       prev.setNext(node);
       numElts++;
   }
}
```



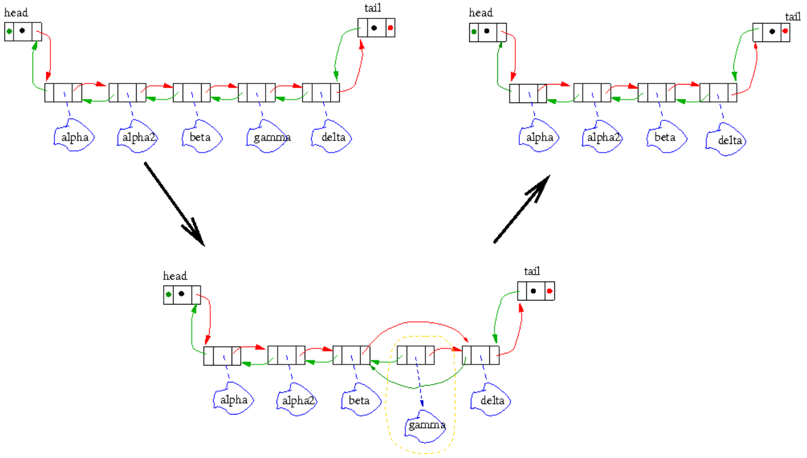
Method add cont'd

```
public void add(int inx, EltType newElt)
{  if (inx == size()){ . . . }
   else
   {   LLNode<EltType> next = nodeAtIndex(inx);
       LLNode<EltType> prev = next.getPrev();
       LLNode<EltType> node =
           new LLNode<EltType>(prev, next, newElt);
       next.setPrev(node); prev.setNext(node);
       numElts++;
   }
}
```



Method remove

Operation: remove(3)

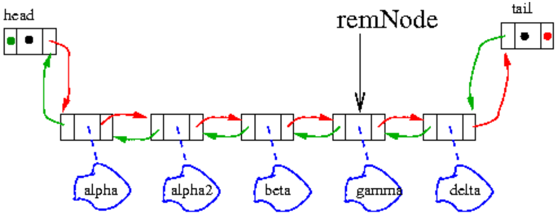


Method remove

```
private EltType removeNode(LLNode<EltType> remNode)
{  LLNode<EltType> next = remNode.getNext();
   LLNode<EltType> prev = remNode.getPrev();
   prev.setNext(next);  next.setPrev(prev);
   numElts--;
   return remNode.element();
}
public EltType remove(int inx)
{  LLNode<EltType> node;
   node = nodeAtIndex(inx);
   return removeNode(node);
}
```

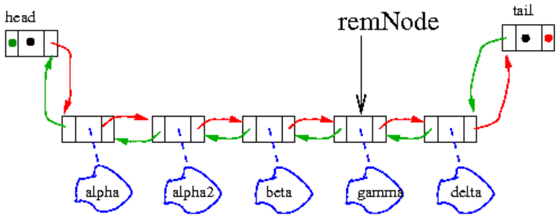
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{   LLNode<EltType> next = remNode.getNext();
    LLNode<EltType> prev = remNode.getPrev();
    prev.setNext(next);
      next.setPrev(prev);
    numElts--;
    return remNode.element();
}
```



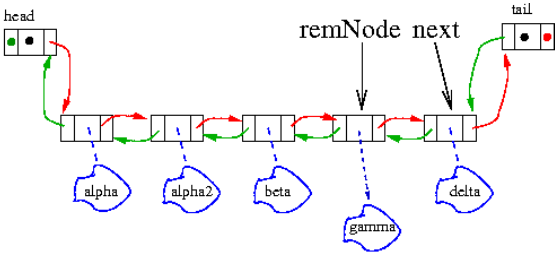
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{   LLNode<EltType> next = remNode.getNext();
    LLNode<EltType> prev = remNode.getPrev();
    prev.setNext(next);
       next.setPrev(prev);
    numElts--;
    return remNode.element();
}
```



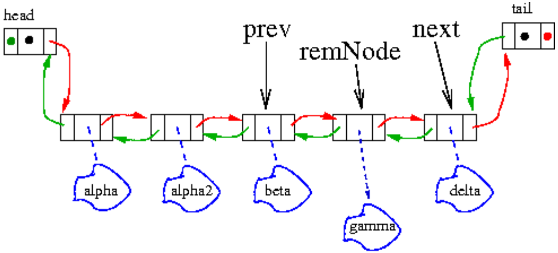
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{  LLNode<EltType> next = remNode.getNext();
   LLNode<EltType> prev = remNode.getPrev();
   prev.setNext(next);
     next.setPrev(prev);
   numElts--;
   return remNode.element();
}
```



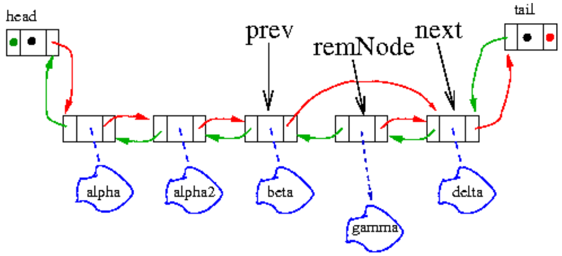
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{   LLNode<EltType> next = remNode.getNext();
    LLNode<EltType> prev = remNode.getPrev();
    prev.setNext(next);
      next.setPrev(prev);
    numElts--;
    return remNode.element();
}
```



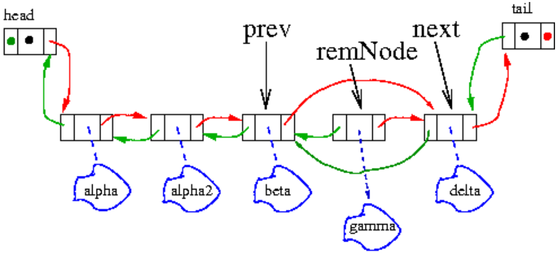
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{   LLNode<EltType> next = remNode.getNext();
    LLNode<EltType> prev = remNode.getPrev();
    prev.setNext(next);
       next.setPrev(prev);
    numElts--;
    return remNode.element();
}
```



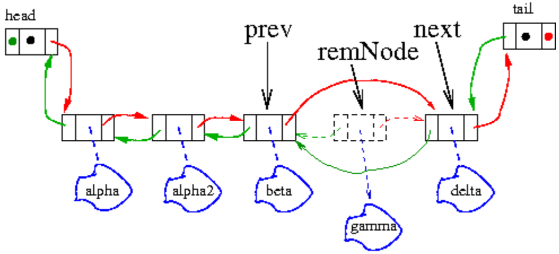
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{   LLNode<EltType> next = remNode.getNext();
    LLNode<EltType> prev = remNode.getPrev();
    prev.setNext(next);
      next.setPrev(prev);
    numElts--;
    return remNode.element();
}
```



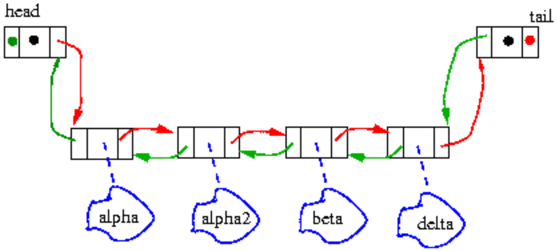
Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{  LLNode<EltType> next = remNode.getNext();
   LLNode<EltType> prev = remNode.getPrev();
   prev.setNext(next);
     next.setPrev(prev);
   numElts--;
   return remNode.element();
}
```



Method remove cont'd

```
private EltType removeNode(LLNode<EltType> remNode)
{  LLNode<EltType> next = remNode.getNext();
   LLNode<EltType> prev = remNode.getPrev();
   prev.setNext(next);
     next.setPrev(prev);
   numElts--;
   return remNode.element();
}
```



(Tidied up to omit next, prev and remNode)