Data Structure

Lecture#7: Lists, Stacks, and Queues (Chapter 4)

U Kang
Seoul National University

In This Lecture

- Learn the List data structure
- Compare the array-based and link-based implementation of List in terms of time and space
- Understand the motivation and the main idea of freelist

Lists

- A list is a finite, ordered sequence of data items.
- Important concept: List elements have a position.
- Notation: $<a_0, a_1, \ldots, a_{n\text{-}1}>$
- What operations should we implement?

List Implementation Concepts

- Our list implementation will support the concept of a current position.
- Operations will act relative to the current position.

<20, 23 | 12, 15>

List ADT

```
public interface List<E> {
  public void clear();
  public void insert(E item);
  public void append(E item);
  public E remove();
  public void moveToStart();
  public void moveToEnd();
  public void prev();
  public void next();
  public int length();
  public int currPos();
  public void moveToPos(int pos);
  public E getValue();
}
```

List ADT Examples

- List: <12 | 32, 15>

```
L.insert(99);
```

Result: <12 | 99, 32, 15>

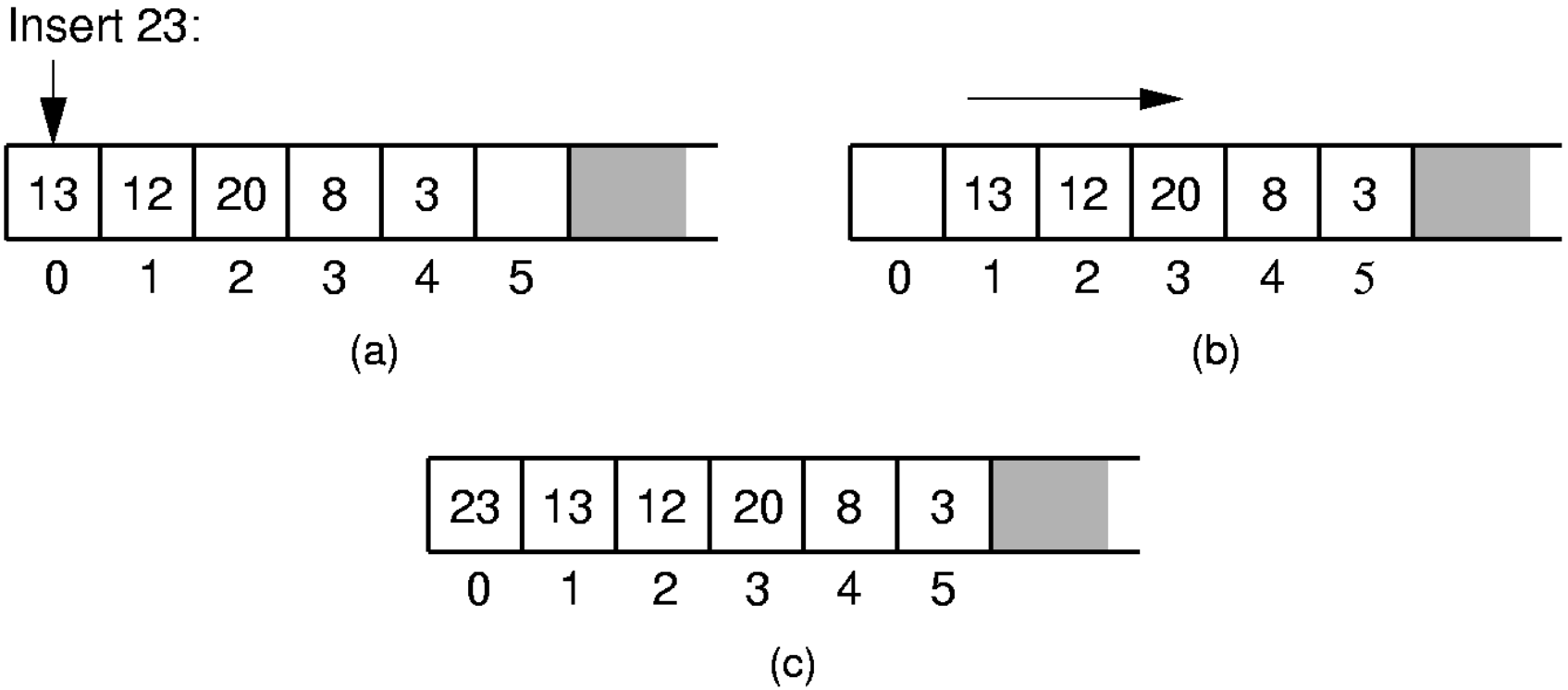
- Iterate through the whole list:

```
for (L.moveToStart(); L.currPos()<L.length();
     L.next()) {
  it = L.getValue();
  doSomething(it);
}
```

List Find Function

```
/** @return True if k is in list L,
    false otherwise */
public static boolean
find(List<Integer> L, int k) {
  for (L.moveToStart();
       L.currPos()<L.length(); L.next())
    if (k == L.getValue()) return true;
  return false;              // k not found
}
```

Array-Based List Insert



Array-Based List Class (1)

```
class AList<E> implements List<E> {
  private static final int defaultSize = 10;

  private int maxSize;
  private int listSize;
  private int curr;
  private E[] listArray;

  // Constructors
  AList() { this(defaultSize); }
  AList(int size) {
    maxSize = size;
    listSize = curr = 0;
    listArray = (E[])new Object[size];
  }
```

Array-Based List Class (2)

```
public void clear()
  { listSize = curr = 0; }
public void moveToStart() { curr = 0; }
public void moveToEnd() { curr = listSize; }
public void prev() { if (curr != 0) curr--; }
public void next()
  { if (curr < listSize) curr++; }
public int length() { return listSize; }
public int currPos() { return curr; }
```

Array-Based List Class (3)

```
public void moveToPos(int pos) {
  assert (pos>=0) && (pos<=listSize) :
          "Position out of range";
  curr = pos;
}

public E getValue() {
  assert (curr >= 0) && (curr < listSize) :
          "No current element";
  return listArray[curr];
}
```

Insert

```
/** Insert "it" at current position */
public void insert(E it) {
  assert listSize < maxSize :
         "List capacity exceeded";
  for (int i=listSize; i>curr; i--)
    listArray[i] = listArray[i-1];
  listArray[curr] = it;
  listSize++;
}
```

Append

```
public void append(E it) { // Append "it"
  assert listSize < maxSize :
          "List capacity exceeded";
  listArray[listSize++] = it;
}
```

Remove

```
/** Remove and return the current element */
public E remove() {
  if ((curr < 0) || (curr >= listSize))
    return null;
  E it = listArray[curr];
  for(int i=curr; i<listSize-1; i++)
    listArray[i] = listArray[i+1];
  listSize--;
  return it;
}
```

Array Based List

- Strengths?

- Weaknesses?

Link Class

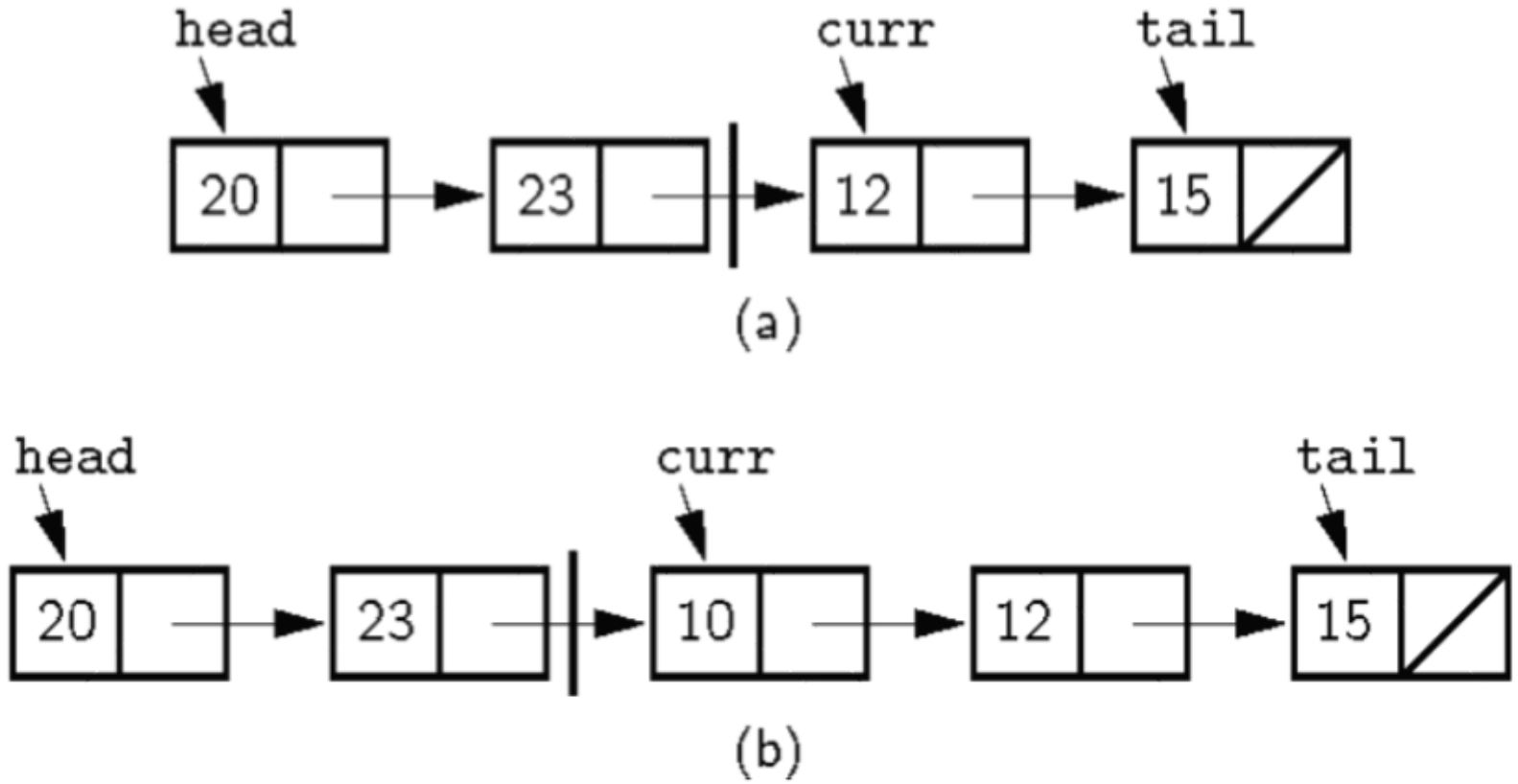
- Dynamic allocation of new list elements.

```
class Link<E> {
  private E element;
  private Link<E> next;

  // Constructors
  Link(E it, Link<E> nextval)
    { element = it;  next = nextval; }
  Link(Link<E> nextval) { next = nextval; }

  Link<E> next() { return next; }
  Link<E> setNext(Link<E> nextval)
    { return next = nextval; }
  E element() { return element; }
  E setElement(E it) { return element = it; }
}
```

Linked List Position (1)

Naïve Method

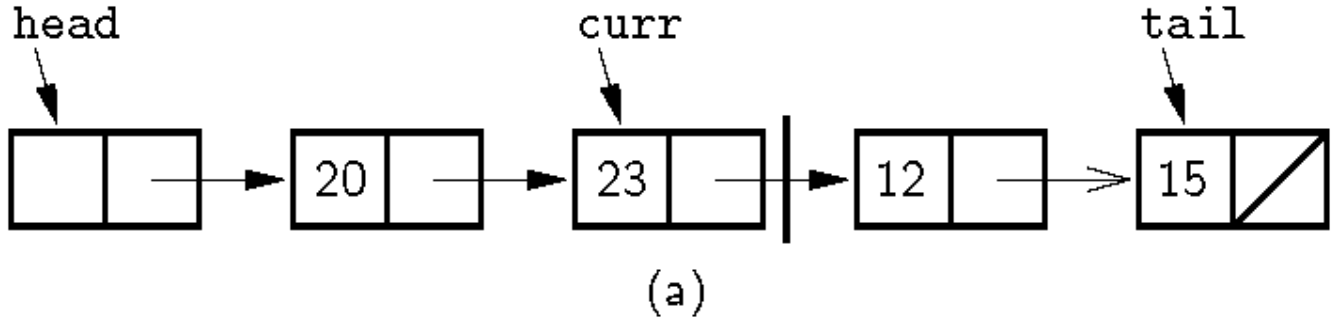
Insert 10 at the current position



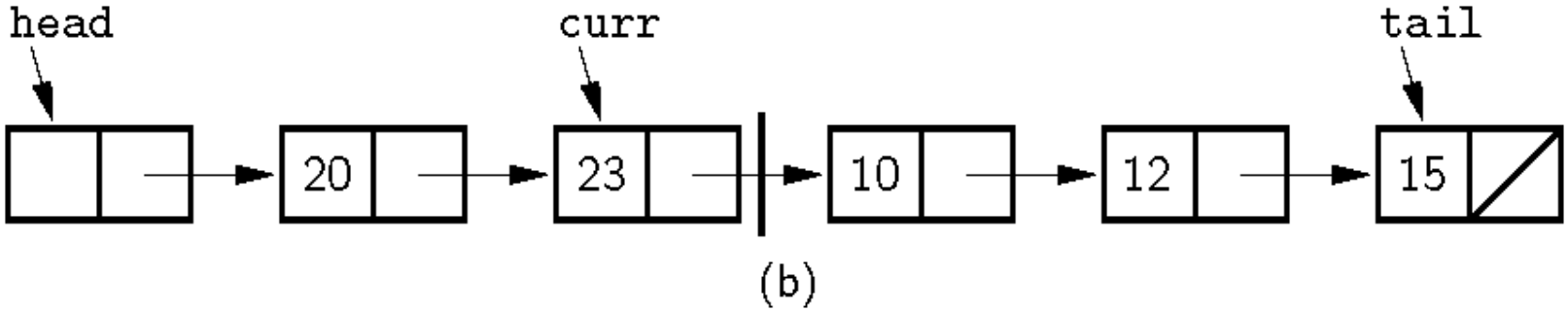
Linked List Position (2)

Improved Method

Insert 10 at the current position



(a)



(b)

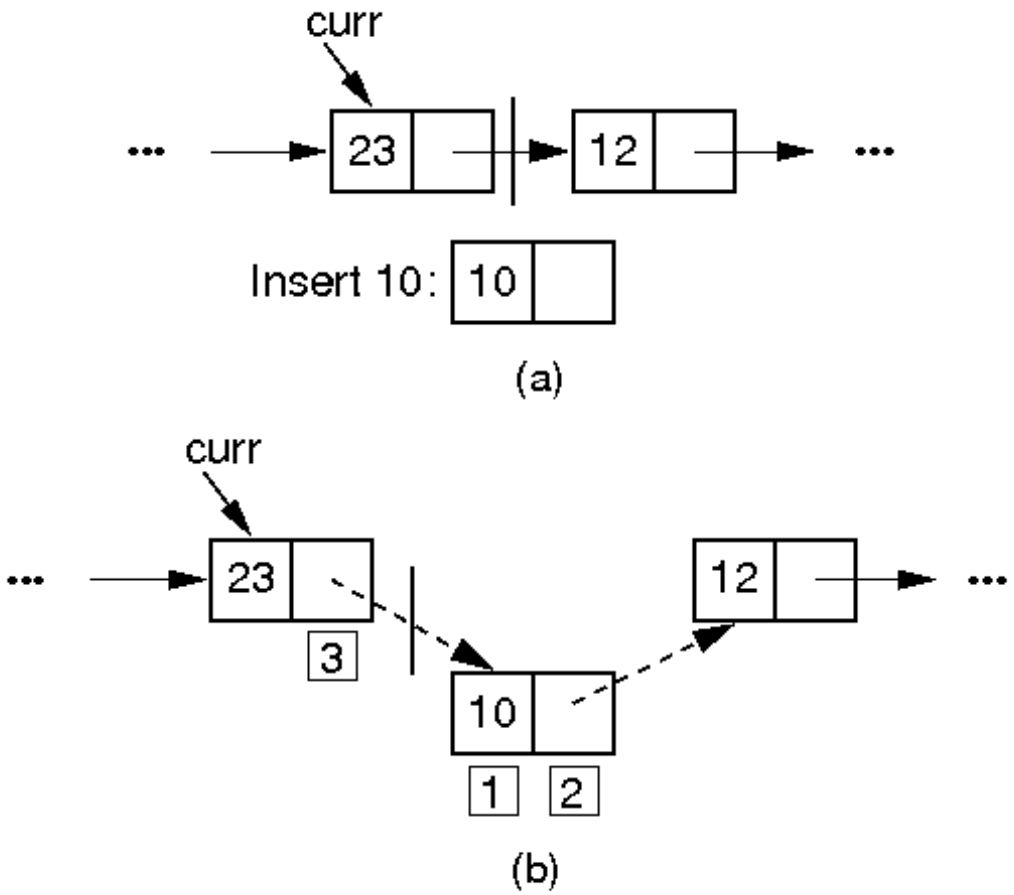
Linked List Class (1)

```
class LList<E> implements List<E> {
  private Link<E> head;
  private Link<E> tail;
  protected Link<E> curr;
  int cnt;

  //Constructors
  LList(int size) { this(); }
  LList() {
    curr = tail = head = new Link<E>(null);
    cnt = 0;
  }
```

Linked List Class (2)

```
public void clear() {
  head.setNext(null);
  curr = tail = head = new Link<E>(null);
  cnt = 0;
}
public void moveToStart() { curr = head; }
public void moveToEnd() { curr = tail; }
public int length() { return cnt; }
public void next() {
  if (curr != tail) { curr = curr.next(); }
}
public E getValue() {
  assert curr.next() != null :
    "Nothing to get";
  return curr.next().element();
}
```

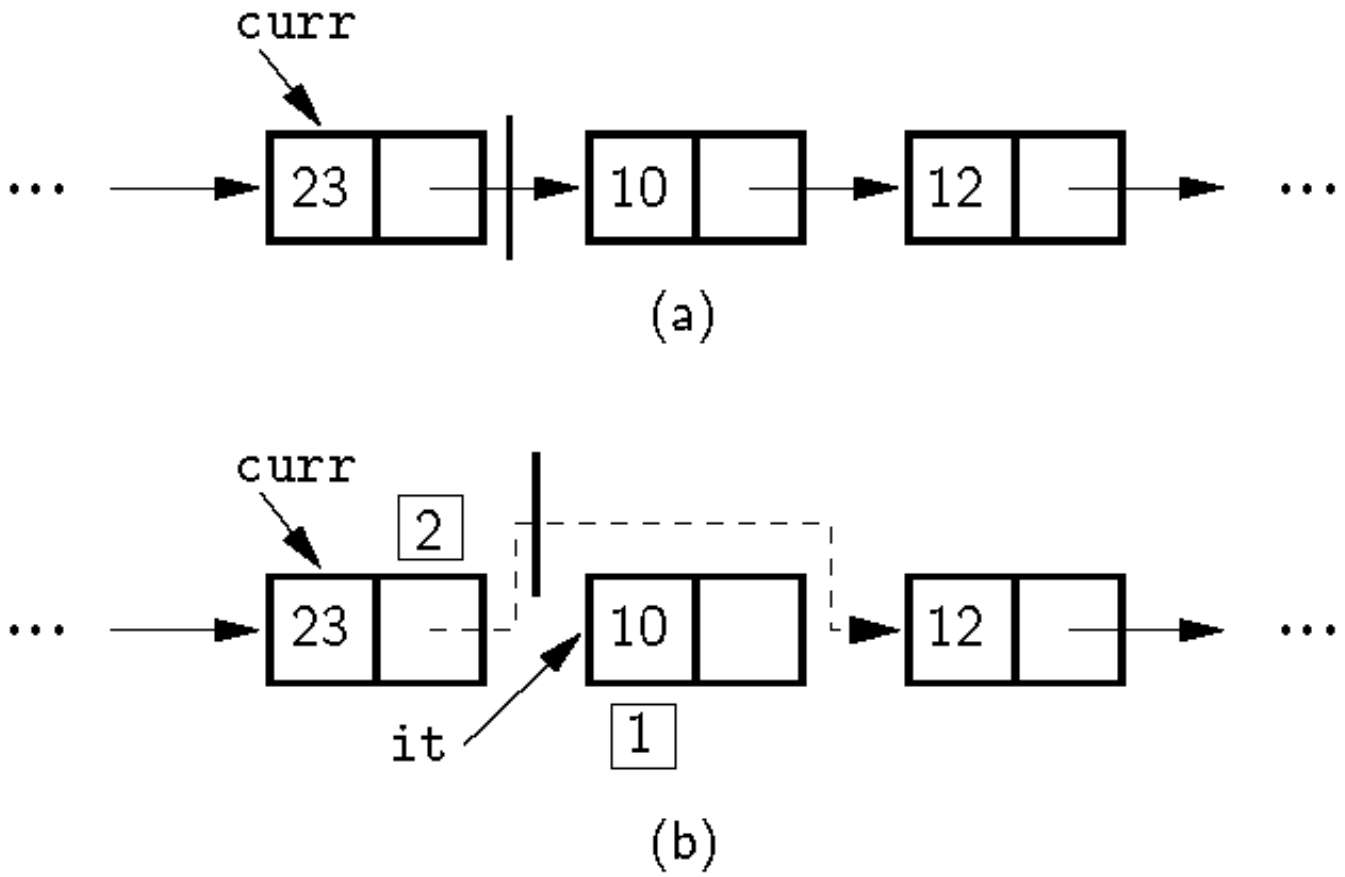
Insertion



Insert/Append

```
// Insert "it" at current position
public void insert(E it) {
  curr.setNext(new Link<E>(it, curr.next()));
  if (tail == curr) tail = curr.next();
  cnt++;
}

public void append(E it) {
  tail = tail.setNext(new Link<E>(it, null));
  cnt++;
}
```

Removal



Remove

```
/** Remove and return current element */
public E remove() {
  if (curr.next() == null) return null;
  E it = curr.next().element();
  if (tail == curr.next()) tail = curr;
  curr.setNext(curr.next().next());
  cnt--;
  return it;
}
```

Prev

```
/** Move curr one step left;
    no change if already at front */
public void prev() {
  if (curr == head) return;
  Link<E> temp = head;
  // March down list until we find the
  // previous element
  while (temp.next() != curr)
    temp = temp.next();
  curr = temp;
}
```

Get/Set Position

```
/** Return position of the current element */
public int currPos() {
  Link<E> temp = head;
  int i;
  for (i=0; curr != temp; i++)
    temp = temp.next();
  return i;
}

/** Move down list to "pos" position */
public void moveToPos(int pos) {
  assert (pos>=0) && (pos<=cnt) :
         "Position out of range";
  curr = head;
  for(int i=0; i<pos; i++)
    curr = curr.next();
}
```

Comparison of Implementations

	Array Based List	Linked List
Insertion and Deletion		
Prev / access to an element		
Pre-allocate space		
Additional space overhead		

Space Comparison

- "Break-even" point:

$DE = n(P + E)$;

$$n = \frac{DE}{P + E}$$

D: Number of maximum elements in array.
n: Number of elements in linked list
E: Space for data value.
P: Space for pointer.

- When should we use array to minimize space?

Freelists

System **new** and garbage collection are slow.

- Add freelist support to the Link class.

Link Class Extensions

```
static Link freelist = null;

static <E> Link<E> get(E it, Link<E> nextval) {
  if (freelist == null)
    return new Link<E>(it, nextval);
  Link<E> temp = freelist;
  freelist = freelist.next();
  temp.setElement(it);
  temp.setNext(nextval);
  return temp;
}

void release() {  // Return to freelist
  element = null;
  next = freelist;
  freelist = this;
}
```

Not Using Freelist

```
public void insert(E it) {
  curr.setNext(new Link<E>(it, curr.next()));
  if (tail == curr) tail = curr.next();
  cnt++;
}

public E remove() {
  if (curr.next() == null) return null;
  E it = curr.next().element();
  if (tail == curr.next()) tail = curr;
  curr.setNext(curr.next().next());
  cnt--;
  return it;
}
```

Using Freelist

```
public void insert(E it) {
  curr.setNext(Link.get(it, curr.next()));
  if (tail == curr) tail = curr.next();
  cnt++;
}

public E remove() {
  if (curr.next() == null) return null;
  E it = curr.next().element();
  if (tail == curr.next()) tail = curr;
  Link<E> tempptr = curr.next();
  curr.setNext(curr.next().next());
  tempptr.release();
  cnt--;
  return it;
}
```

What you need to know

- ADT of the List data structure and when it is needed
- Comparison of the array-based and link-based implementations of List in terms of time and space
- The motivation and the main idea of freelist

Questions?