### CS61B Lecture #4: Simple Pointer Manipulation

**Recreation** Prove that for every acute angle  $\alpha > 0$ ,

 $\tan \alpha + \cot \alpha \ge 2$ 

#### Announcements

- Today: More pointer hacking.
- Handing in labs and homework: We'll be lenient about accepting late homework and labs for lab1, lab2, and hwO. Just get it done: part of the point is getting to understand the tools involved. We will not accept submissions by email.
- We will feel free to interpret the absence of a central repository for you or a lack of a lab1 submission from you as indicating that you intend to drop the course.
- Project 0 to be released tonight.
- HW1 is released.

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 1

### Small Test of Understanding

- In Java, the keyword final in a variable declaration means that the variable's value may not be changed after the variable is initialized.
- Is the following class valid?

```
public class Issue {
     private final IntList aList = new IntList(0, null);
     public void modify(int k) {
          this.aList.head = k;
```

Why or why not?

Last modified: Fri Sep 6 15:32:48 2019 CS61B: Lecture #4 2

# Small Test of Understanding

- In Java, the keyword final in a variable declaration means that the variable's value may not be changed after the variable is initialized.
- Is the following class valid?

```
public class Issue {
    private final IntList aList = new IntList(0, null);
     public void modify(int k) {
          this.aList.head = k;
```

Why or why not?

Answer: This is valid. Although modify changes the head variable of the object pointed to by aList, it does not modify the contents of aList itself (which is a pointer).

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 3

### Destructive Incrementing

Destructive solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                 X = IntList.list(3, 43, 56):
  if (P == null)
                                                 /* IntList.list from HW #1 */
    return null:
                                                 Q = dincrList(X, 2);
  else {
    P.head += n;
    P.tail = dincrList(P.tail, n);
    return P;
 }
                                                            3
/** Destructively add N to L's items. */
static IntList dincrList(IntList L. int n)
  // 'for' can do more than count!
  for (IntList p = L; p != null; p = p.tail)
   p.head += n;
  return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                              CS61B: Lecture #4 4
```

### Destructive Incrementing

Destructive solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                 X = IntList.list(3, 43, 56);
  if (P == null)
                                                 /* IntList.list from HW #1 */
   return null;
                                                 Q = dincrList(X, 2);
  else {
   P.head += n;
                                              X: -
   P.tail = dincrList(P.tail, n);
    return P;
/** Destructively add N to L's items. */
static IntList dincrList(IntList L, int n)
  // 'for' can do more than count!
  for (IntList p = L; p != null; p = p.tail)
   p.head += n;
  return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                              CS61B: Lecture #4 5
```

### Destructive Incrementing

Destructive solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                 X = IntList.list(3, 43, 56);
  if (P == null)
                                                 /* IntList.list from HW #1 */
    return null;
                                                 Q = dincrList(X, 2);
  else {
    P.head += n;
                                               X: | -
    P.tail = dincrList(P.tail, n);
    return P;
                                                            5
/** Destructively add N to L's items. */
static IntList dincrList(IntList L, int n)
  // 'for' can do more than count!
  for (IntList p = L; p != null; p = p.tail)
    p.head += n;
  return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                              CS61B: Lecture #4 6
```

### Destructive Incrementing

Destructive solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                X = IntList.list(3, 43, 56);
 if (P == null)
                                                /* IntList.list from HW #1 */
   return null;
                                                Q = dincrList(X, 2);
 else {
   P.head += n:
   P.tail = dincrList(P.tail, n);
   return P;
                                                          5
                                                                              -56
/** Destructively add N to L's items. */
static IntList dincrList(IntList L, int n)
 // 'for' can do more than count!
 for (IntList p = L; p != null; p = p.tail)
   p.head += n;
 return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                             CS61B: Lecture #4 7
```

### **Destructive Incrementing**

Destructive solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                X = IntList.list(3, 43, 56);
  if (P == null)
                                                /* IntList.list from HW #1 */
    return null;
                                                Q = dincrList(X, 2);
  else {
   P.head += n:
                                              x:ြ
    P.tail = dincrList(P.tail, n);
    return P;
                                                           5
                                                                    45
/** Destructively add N to L's items. */
static IntList dincrList(IntList L, int n)
  // 'for' can do more than count!
  for (IntList p = L; p != null; p = p.tail)
    p.head += n;
  return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                             CS61B: Lecture #4 8
```

### Destructive Incrementing

Destructive solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                X = IntList.list(3, 43, 56):
 if (P == null)
                                                /* IntList.list from HW #1 */
   return null:
                                                Q = dincrList(X, 2);
 else {
   P.head += n;
   P.tail = dincrList(P.tail, n);
   return P;
 }
                                                          151 -
                                                                    45 -
                                                                              -58
/** Destructively add N to L's items. */
static IntList dincrList(IntList L. int n)
                                              P: [-]
 // 'for' can do more than count!
 for (IntList p = L; p != null; p = p.tail)
   p.head += n:
 return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                             CS61B: Lecture #4 9
```

# Destructive Incrementing

 ${\it Destructive}$  solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. */
static IntList dincrList(IntList P, int n) {
                                                X = IntList.list(3, 43, 56):
  if (P == null)
                                                 /* IntList.list from HW #1 */
   return null:
                                                 Q = dincrList(X, 2);
  else {
   P.head += n;
                                              X: | -
    P.tail = dincrList(P.tail, n);
    return P;
 }
                                                           5
                                                                     45
/** Destructively add N to L's items. */
static IntList dincrList(IntList L. int n)
                                               P: \
  // 'for' can do more than count!
  for (IntList p = L; p != null; p = p.tail)
   p.head += n;
  return L;
Last modified: Fri Sep 6 15:32:48 2019
                                                             CS61B: Lecture #4 10
```

### Another Example: Non-destructive List Deletion

If L is the list [2, 1, 2, 9, 2], we want removeAll(L,2) to be the new list [1, 9].

```
/** The list resulting from removing all instances of X from L
 * non-destructively. */
static IntList removeAll(IntList L, int x) {
   if (L == null)
      return /*( null with all x's removed )*/;
   else if (L.head == x)
      return /*( L with all x's removed (L!=null, L.head==x) )*/;
   else
      return /*( L with all x's removed (L!=null, L.head!=x) )*/;
}
```

### Another Example: Non-destructive List Deletion

If L is the list [2, 1, 2, 9, 2], we want removeAll(L, 2) to be the new list [1, 9].

```
/** The list resulting from removing all instances of X from L
 * non-destructively. */
static IntList removeAll(IntList L, int x) {
  if (L == null)
    return null;
  else if (L.head == x)
    return /*( L with all x's removed (L!=null, L.head==x) )*/;
  else
    return /*( L with all x's removed (L!=null, L.head!=x) )*/;
```

### Another Example: Non-destructive List Deletion

```
If L is the list [2, 1, 2, 9, 2], we want removeAll(L,2) to be the new
list [1, 9].

/** The list resulting from removing all instances of X from L
 * non-destructively. */
static IntList removeAll(IntList L, int x) {
  if (L == null)
    return null;
  else if (L.head == x)
    return removeAll(L.tail, x);
  else
    return /*( L with all x's removed (L!=null, L.head!=x) )*/;
}
```

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 13

### Another Example: Non-destructive List Deletion

```
If L is the list [2, 1, 2, 9, 2], we want removeAll(L,2) to be the new list [1, 9].
```

```
/** The list resulting from removing all instances of X from L
* non-destructively. */
static IntList removeAll(IntList L, int x) {
   if (L == null)
      return null;
   else if (L.head == x)
      return removeAll(L.tail, x);
   else
      return new IntList(L.head, removeAll(L.tail, x));
}
```

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 14

### Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
  * of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
  IntList result, last;
  result = last = null;
  for (; L != null; L = L.tail) {
    if (x == L.head)
      continue;
    else if (last == null)
      result = last = new IntList(L.head, null);
    else
      last = last.tail = new IntList(L.head, null);
}
return result;
}
```

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 15

# Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
* of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last;
                                        P: 🗀
 result = last = null;
 for ( ; L != null; L = L.tail) {
   if (x == L.head)
                                   result:
     continue:
   else if (last == null)
                                     last: 
abla
                                                    removeAll (P, 2)
     result = last = new IntList(L.head, null);
     last = last.tail = new IntList(L.head, null);
 return result;
```

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 16

# Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
* of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last;
                                                 <del>-</del>2]
 result = last = null;
                                        L: T
 for ( ; L != null; L = L.tail) {
   if (x == L.head)
                                   result: N
     continue;
   else if (last == null)
                                                    removeAll (P, 2)
                                     last: 🖯
     result = last = new IntList(L.head, null);
                                                    P does not change!
   else
     last = last.tail = new IntList(L.head, null);
 return result;
```

### Last modified: Fri Sep 6 15:32:48 2019 CS61B: Lecture #4 17

### Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
 * of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last:
                                      P: -2 - 2 - 2 -
 result = last = null;
                                      L: []
 for ( ; L != null; L = L.tail) {
   if (x == L.head)
                                  result: 🗀
                                                1\mathbb{N}
     continue;
   else if (last == null)
                                    last:
                                                  removeAll (P, 2)
     result = last = new IntList(L.head, null);
                                                  P does not change!
   else
     last = last.tail = new IntList(L.head, null);
 return result;
```

Last modified: Fri Sep 6 15:32:48 2019

CS61B: Lecture #4 18

#### Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
* of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last;
                                        P: 🗀
                                               <del>-</del>2<del>1 -</del>11 <del>-</del>21 -9N
 result = last = null;
                                        L: T
 for ( ; L != null; L = L.tail) {
   if (x == L.head)
                                   result:
     continue:
   else if (last == null)
                                     last: 🗖
                                                     removeAll (P, 2)
     result = last = new IntList(L.head, null);
                                                    P does not change!
     last = last.tail = new IntList(L.head, null);
 return result;
```

CS61B: Lecture #4 19

#### Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
 * of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last;
                                          P: [ <del>-</del>
                                                 <del>-</del>21<del>3 -</del>11<del>3 -</del>21<del>3 -</del>9N
  result = last = null;
                                          に日
  for ( ; L != null; L = L.tail) {
   if (x == L.head)
                                     result:
                                                    1
     continue:
    else if (last == null)
                                                       removeAll (P, 2)
                                       last: 🗔
     result = last = new IntList(L.head, null);
                                                       P does not change!
      last = last.tail = new IntList(L.head, null);
  return result;
```

Last modified: Fri Sep 6 15:32:48 2019 CS61B: Lecture #4 20

#### Iterative Non-destructive List Deletion

Last modified: Fri Sep 6 15:32:48 2019

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
* of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last;
                                    P: - 2 - 1 - 2 - 9 N
 result = last = null;
 for ( ; L != null; L = L.tail) {
                                    L: T
   if (x == L.head)
                                result: 1
    continue:
   else if (last == null)
                                               removeAll (P, 2)
    result = last = new IntList(L.head, null);
                                               P does not change!
     last = last.tail = new IntList(L.head, null);
 return result;
```

Last modified: Fri Sep 6 15:32:48 2019 CS61B: Lecture #4 21

## Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```
/** The list resulting from removing all instances
* of X from L non-destructively. */
static IntList removeAll(IntList L, int x) {
 IntList result, last;
                                     P: -2 -1 -9 \
 result = last = null;
 for ( ; L != null; L = L.tail) {
                                     L: 
abla
   if (x == L.head)
                                 result:
                                             <del>-</del>11-
    continue:
                                                removeAll (P, 2)
   else if (last == null)
                                   last: 🕞
    result = last = new IntList(L.head, null);
                                                P does not change!
     last = last.tail = new IntList(L.head, null);
 return result;
```

Last modified: Fri Sep 6 15:32:48 2019 CS61B: Lecture #4 22

### Destructive Deletion

```
- : Original
                                ....: : after Q = dremoveAll (Q,1)
Q: - 1 - 2 - 3 - 1 - 1 - 0 - 1 \
/** The list resulting from removing all instances of X from L.
* The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
  if (L == null)
    return /*( null with all x's removed )*/;
  else if (L.head == x)
    return /*( L with all x's removed (L != null) )*/;
  else {
     /*{ Remove all x's from L's tail. }*/;
     return L:
Last modified: Fri Sep 6 15:32:48 2019
                                                  CS61B: Lecture #4 23
```

### **Destructive Deletion**

```
- : Original
                                   ---- : after Q = dremoveAll (Q,1)
                →2] <del>-</del>
                                   1 - 1 - 0
                                                             1 
/** The list resulting from removing all instances of X from L.
 * The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
 if (L == null)
     return /*( null with all x's removed )*/;
  else if (L.head == x)
     return /*( L with all x's removed (L != null) )*/;
  else {
     /*{ Remove all x's from L's tail. }*/;
     return L:
Last modified: Fri Sep 6 15:32:48 2019
                                                     CS61B: Lecture #4 24
```

#### **Destructive Deletion**

```
- : Original
                                     .... : after Q = dremoveAll (Q,1)
                 →2 →
                          →3 <del>-</del>.
                                   <del>-</del>11 <del>-</del>11 -
                                                     →0 <del>3 →</del>1 \
/** The list resulting from removing all instances of X from L.
* The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
  if (L == null)
    return /*( null with all x's removed )*/;
  else if (L.head == x)
    return /*( L with all x's removed (L != null) )*/;
     /*{ Remove all x's from L's tail. }*/;
     return L;
  }
Last modified: Fri Sep 6 15:32:48 2019
                                                        CS61B: Lecture #4 25
```

#### Destructive Deletion

```
- : Original
                                 ---- : after Q = dremoveAll (Q,1)
         1 - 2 -
                       <del>-3-1-1-1-1-1</del>
/** The list resulting from removing all instances of X from L.
 * The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
 if (L == null)
    return /*( null with all x's removed )*/;
  else if (L.head == x)
    return /*( L with all x's removed (L != null) )*/;
     /*{ Remove all x's from L's tail. }*/;
     return L;
  }
}
Last modified: Fri Sep 6 15:32:48 2019
                                                   CS61B: Lecture #4 26
```

#### Destructive Deletion

```
/** The list resulting from removing all instances of X from L.

* The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {

if (L == null)
    return null;
else if (L.head == x)
    return /*( L with all x's removed (L != null) )*/;
else {
    /*{ Remove all x's from L's tail. }*/;
    return L;
}

Last modified: Fri Sep 6 15:32:48 2019

C5618: Lecture #4 27
```

#### Destructive Deletion

```
/** The list resulting from removing all instances of X from L.

* The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
   if (L == null)
      return
   else if (L.head == x)
      return dremoveAll(L.tail, x);
   else {
      /*{ Remove all x's from L's tail. }*/;
      return L;
   }
}
```

### **Destructive Deletion**

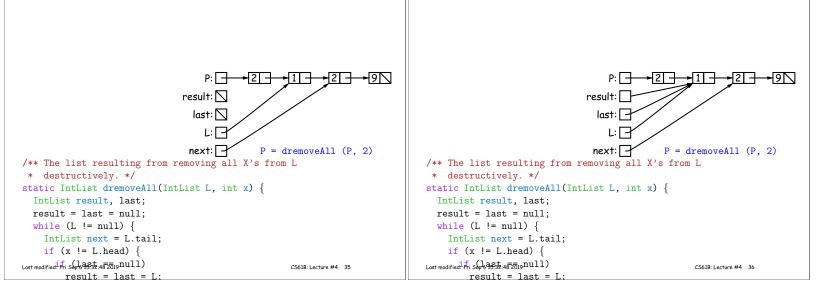
```
- : Original
                                  ....: : after Q = dremoveAll (Q,1)
        <del>-</del>11 <del>-</del>21 <del>-</del>
                        /** The list resulting from removing all instances of X from L.
* The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
 if (L == null)
    return
  else if (L.head == x)
    return dremoveAll(L.tail, x);
  else {
     L.tail = dremoveAll(L.tail, x);
     return L;
  }
Last modified: Fri Sep 6 15:32:48 2019
                                                    CS61B: Lecture #4 29
```

### Iterative Destructive Deletion

```
/** The list resulting from removing all X's from L
* destructively. */
static IntList dremoveAll(IntList L, int x) {
  IntList result, last;
  result = last = null;
  while (L != null) {
    IntList next = L.tail;
    if (x != L.head) {
      if (last == null)
        result = last = L;
      else
        last = last.tail = L;
      L.tail = null;
    L = next;
  return result;
Last modified: Fri Sep 6 15:32:48 2019
                                                      CS61B: Lecture #4 30
```

#### Iterative Destructive Deletion Iterative Destructive Deletion P: - 2 - 9 N result: $\square$ result: N last: last: L: $\square$ next: P = dremoveAll (P, 2) next: P = dremoveAll (P, 2)/\*\* The list resulting from removing all X's from L /\*\* The list resulting from removing all X's from L \* destructively. \*/ \* destructively. \*/ static IntList dremoveAll(IntList L, int x) { static IntList dremoveAll(IntList L, int x) { IntList result, last; IntList result, last; result = last = null; result = last = null; while (L != null) { while (L != null) { IntList next = L.tail; IntList next = L.tail; if (x != L.head) { if (x != L.head) { Last modified Fr Schast: 48 2019 null) Last modified Fri Schlasst 48 2019 null) CS61B: Lecture #4 31 CS61B: Lecture #4 32 result = last = L; result = last = L;

```
Iterative Destructive Deletion
                                                                                                      Iterative Destructive Deletion
                                                       <del>-</del>11<del>--</del>21<del>--</del>9N
                                                                                                                             P: 📑
                                                                                                                                      <del>-</del>21<del>3 -</del>11<del>3 -</del>21<del>3 -</del>9N
                                  result: \
                                                                                                                        result: \
                                                                                                                           last: 🔽
                                    last: \
                                   next:
                                                  P = dremoveAll (P, 2)
                                                                                                                          next:
                                                                                                                                         P = dremoveAll (P, 2)
/** The list resulting from removing all X's from L
                                                                                      /** The list resulting from removing all X's from L
* destructively. */
                                                                                       * destructively. */
static IntList dremoveAll(IntList L, int x) {
                                                                                      static IntList dremoveAll(IntList L, int x) {
 IntList result, last;
                                                                                        IntList result, last;
 result = last = null;
                                                                                        result = last = null;
 while (L != null) {
                                                                                        while (L != null) {
    IntList next = L.tail;
                                                                                          IntList next = L.tail;
    if (x != L.head) {
                                                                                           if (x != L.head) {
Last modified Fri Sep 6 15:35:45 2019 null)
                                                                                      Last modified Fri Septenst: 48 2019 null)
                                                         CS61B: Lecture #4 33
                                                                                                                                                CS61B: Lecture #4 34
        result = last = L:
                                                                                               result = last = L:
```



Iterative Destructive Deletion

Iterative Destructive Deletion

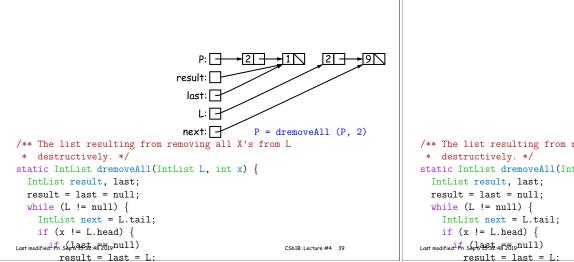
# 21<del>---</del>9N result: [ last: -L: [next: P = dremoveAll (P, 2) /\*\* The list resulting from removing all X's from L \* destructively. \*/ static IntList dremoveAll(IntList L, int x) { IntList result, last; result = last = null; while (L != null) { IntList next = L.tail; if (x != L.head) { Last modified Fr Ser (1953 t. 48 2019 null) CS61B: Lecture #4 37 result = last = L;

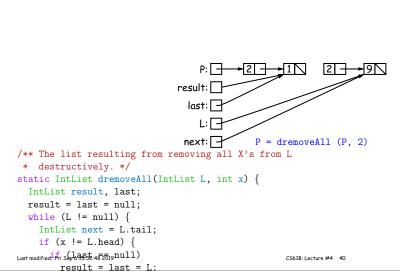
Iterative Destructive Deletion

Iterative Destructive Deletion

```
21<del>--</del>9N
                                 result: [
                                   last:
                                     L: [-
                                  next: -
                                                 P = dremoveAll (P, 2)
/** The list resulting from removing all X's from L
 * destructively. */
static IntList dremoveAll(IntList L, int x) {
  IntList result, last;
  result = last = null;
  while (L != null) {
    IntList next = L.tail;
    if (x != L.head) {
Last modified Fri Schlasst 48 2019 null)
                                                       CS61B: Lecture #4 38
        result = last = L:
```

Iterative Destructive Deletion

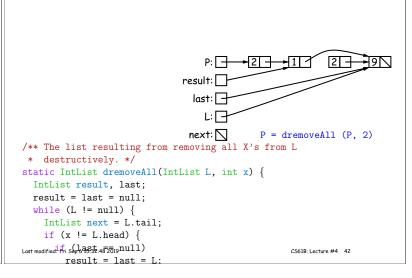




Iterative Destructive Deletion

result = last = L;

Iterative Destructive Deletion



Iterative Destructive Deletion

#### Iterative Destructive Deletion

```
result:
                                   last:
                                     L:N
                                  next:
                                                 P = dremoveAll (P, 2)
/** The list resulting from removing all X's from L
 * destructively. */
static IntList dremoveAll(IntList L, int x) {
  IntList result, last;
  result = last = null;
  while (L != null) {
    IntList next = L.tail;
    if (x != L.head) {
Last modified Fr Ser (1953 t. 48 2019 null)
                                                       CS61B: Lecture #4 43
        result = last = L;
```

#### Iterative Destructive Deletion

```
result:
                                  last:
                                     L: 🛛
                                  next:
                                                P = dremoveAll (P, 2)
/** The list resulting from removing all X's from L
 * destructively. */
static IntList dremoveAll(IntList L, int x) {
  IntList result, last;
  result = last = null;
  while (L != null) {
    IntList next = L.tail;
    if (x != L.head) {
Last modified Fri Sept 3532:48 2019 null)
                                                      CS61B: Lecture #4 44
        result = last = L:
```

## Aside: How to Write a Loop (in Theory)

- Try to give a description of how things look on any arbitrary iteration of the loop.
- This description is known as a *loop invariant*, because it is always true at the start of each iteration.
- The loop body then must
  - Start from any situation consistent with the invariant;
  - Make progress in such a way as to make the invariant true again.

```
// Invariant must be true here
while (condition) { // condition must not have side-effects.
    // (Invariant will necessarily be true here.)
    loop body
    // Invariant must again be true here
}
// Invariant true and condition false.
```

• So if our loop gets the desired answer whenever *Invariant* is true and *condition* false, our job is done!

## Relationship to Recursion

 Another way to see this is to consider an equivalent recursive procedure:

```
/** Assuming Invariant, produce a situation where Inveriant
* is true and condition is false. */
void loop() {
    // Invariant assumed true here.
    if (condition) {
        loop body
        // Invariant must be true here.
        loop()
        // Invariant true here and condition false.
    }
}
```

- ullet Here, the invariant is the precondition of the function loop.
- The loop maintains the invariant while making the condition false.
- Idea is to arrange that our actual goal is implied by this post-condition.

Last modified: Fri Sep 6 15:32:48 2019 CS61B: Lecture #4 46

## Example: Loop Invariant for dremoveAll

```
/** The list resulting from removing all X's from L
 * destructively. */
static IntList dremoveAll(IntList L, int x) {
  IntList result, last;
                                                                               -∮9\\
                                          P: -
  result = last = null;
                                    result:
  while ** (L != null) {
    IntList next = L.tail;
                                       last:
    if (x != L.head) {
      if (last == null)
        result = last = L;
                                                       P = dremoveAll (P, 2)
                                    ** Invariant:
        last = last.tail = L;
      L.tail = null:
                                     • result points to the list of items in the
                                       final result except for those from L on-
    L = next:
                                     • L points to an unchanged tail of the
  return result;
                                       original list of items in L.
                                     • last points to the last item in result
                                       or is null if result is null.
Last modified: Fri Sep 6 15:32:48 2019
                                                              CS61B: Lecture #4 47
```