Small Test of Understanding

keyword final in a variable declaration means that the ue may not be changed after the variable is initialized.

ring class valid?

```
s Issue {
e final IntList aList = new IntList(0, null);
void modify(int k) {
his.aList.head = k;

not?
```

ecture #4: Simple Pointer Manipulation

```
bve that for every acute angle \alpha>0,
```

 $\tan \alpha + \cot \alpha > 2$

pointer hacking.

labs and homework: We'll be lenient about accepting rk and labs for lab1, lab2, and hw0. Just get it done: oint is getting to understand the tools involved. We will ubmissions by email.

free to interpret the absence of a central repository ack of a lab1 submission from you as indicating that you up the course.

be released tonight.

sed.

2:48 2019 CS61B: Lecture #4 1

Destructive Incrementing

utions may modify objects in the original list to save

Small Test of Understanding

keyword **final** in a variable declaration means that the ue may not be changed after the variable is initialized. ing class valid?

```
s Issue {
e final IntList aList = new IntList(0, null);
void modify(int k) {
his.aList.head = k;
```

not?

s is valid. Although modify changes the head variable t pointed to by aList, it does not modify the contents elf (which is a pointer).

2:48 2019 CS61B: Lecture #4 3

Destructive Incrementing

utions may modify objects in the original list to save

Destructive Incrementing

utions may modify objects in the original list to save

Destructive Incrementing

utions may modify objects in the original list to save

```
add N to L's items. */
incrList(IntList P, int n) {
                               X = IntList.list(3, 43, 56);
                               /* IntList.list from HW #1 */
                               Q = dincrList(X, 2);
crList(P.tail, n);
 add N to L's items. */
incrList(IntList L. int n)
 more than count!
= L; p != null; p = p.tail)
:48 2019
                                            CS61B: Lecture #4 8
```

Destructive Incrementing

utions may modify objects in the original list to save

```
add N to L's items. */
incrList(IntList P, int n) {
                               X = IntList.list(3, 43, 56):
                               /* IntList.list from HW #1 */
                               Q = dincrList(X, 2);
crList(P.tail, n);
 add N to L's items. */
incrList(IntList L, int n)
 more than count!
= L; p != null; p = p.tail)
:48 2019
                                           CS61B: Lecture #4 7
```

Destructive Incrementing

utions may modify objects in the original list to save

```
add N to L's items. */
incrList(IntList P. int n) {
                              X = IntList.list(3, 43, 56);
                              /* IntList.list from HW #1 */
                              Q = dincrList(X, 2);
crList(P.tail, n);
 add N to L's items. */
incrList(IntList L. int n)
 more than count!
 = L; p != null; p = p.tail)
2:48 2019
```

CS61B: Lecture #4 10

CS61B: Lecture #4 9

Destructive Incrementing

2:48 2019

utions may modify objects in the original list to save

```
add N to L's items. */
incrList(IntList P, int n) {
                                X = IntList.list(3, 43, 56):
                                /* IntList.list from HW #1 */
                                Q = dincrList(X, 2);
crList(P.tail, n);
                                                   <del>-</del>45 <del>-</del>
 add N to L's items. */
incrList(IntList L, int n)
 more than count!
= L; p != null; p = p.tail)
```

Example: Non-destructive List Deletion

```
[2, 1, 2, 9, 2], we want removeAll(L,2) to be the new
esulting from removing all instances of X from L
```

```
ctively. */
 removeAll(IntList L, int x) {
111;
head == x)
( L with all x's removed (L!=null, L.head==x) )*/;
k( L with all x's removed (L!=null, L.head!=x) )*/:
```

2:48 2019 CS61B: Lecture #4 12

Example: Non-destructive List Deletion

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

```
resulting from removing all instances of X from L
ctively. */
 removeAll(IntList L, int x) {
( null with all x's removed )*/;
( L with all x's removed (L!=null, L.head==x) )*/;
( L with all x's removed (L!=null, L.head!=x) )*/;
```

CS61B: Lecture #4 11

2:48 2019

Example: Non-destructive List Deletion

```
[2, 1, 2, 9, 2], we want removeAll(L,2) to be the new
resulting from removing all instances of X from L
ctively. */
 removeAll(IntList L, int x) {
111;
lead == x)
moveAll(L.tail, x);
w IntList(L.head, removeAll(L.tail, x)):
 :48 2019
                                        CS61B: Lecture #4 14
```

Example: Non-destructive List Deletion

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

```
esulting from removing all instances of X from L
ctively. */
 removeAll(IntList L, int x) {
111;
lead == x)
moveAll(L.tail, x);
( L with all x's removed (L!=null, L.head!=x) )*/;
```

CS61B: Lecture #4 13

:48 2019

ative Non-destructive List Deletion

but use front-to-back iteration rather than recursion.

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
                                             <del>-</del>11 <del>-</del>21 <del>-</del>9N
 null;
ull; L = L.tail) {
ead)
                     result:
t == null)
                        last: \nabla
                                         removeAll (P, 2)
ast = new IntList(L.head, null);
t.tail = new IntList(L.head, null);
2:48 2019
                                                 CS61B: Lecture #4 16
```

ative Non-destructive List Deletion

2:48 2019

but use front-to-back iteration rather than recursion.

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
 last;
null;
ull; L = L.tail) {
ead)
t == null)
ast = new IntList(L.head, null);
t.tail = new IntList(L.head, null);
```

CS61B: Lecture #4 15

ative Non-destructive List Deletion

but use front-to-back iteration rather than recursion.

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
                                             <del>-</del>11 <del>-</del>21 <del>-</del>9N
ull; L = L.tail) {
ead)
                                        removeAll (P, 2)
                       last: 🕞
ast = new IntList(L.head, null);
                                       P does not change!
t.tail = new IntList(L.head, null);
2:48 2019
                                                CS61B: Lecture #4 18
```

ative Non-destructive List Deletion

2:48 2019

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

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
 last;
                              null;
ull; L = L.tail) {
ead)
                 result:
 == null)
                   last: \nabla
                                removeAll (P, 2)
ast = new IntList(L.head, null);
                                P does not change!
t.tail = new IntList(L.head, null);
```

CS61B: Lecture #4 17

ative Non-destructive List Deletion

but use front-to-back iteration rather than recursion.

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
                       P: - 2 - 1 - 2 - 9 N
 null;
ull; L = L.tail) {
ead)
                  result: +
                                   removeAll (P, 2)
t == null)
                    last: 🕞
ast = new IntList(L.head, null);
                                  P does not change!
 .tail = new IntList(L.head, null);
:48 2019
                                         CS61B: Lecture #4 20
```

ative Non-destructive List Deletion

but use front-to-back iteration rather than recursion.

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
 last;
                      P: - - 2 - - 9 N
 null:
ull; L = L.tail) {
ead)
                 result: 1
 == null)
                    last: 🕞
                                 removeAll (P, 2)
ast = new IntList(L.head, null);
                                 P does not change!
 .tail = new IntList(L.head, null);
:48 2019
                                        CS61B: Lecture #4 19
```

ative Non-destructive List Deletion

but use front-to-back iteration rather than recursion.

```
ulting from removing all instances
non-destructively. */
emoveAll(IntList L, int x) {
                     P: - 2 - 9 \
 null;
                     L: N
ull; L = L.tail) {
ead)
                 result: - 1 - 9
                               removeAll (P, 2)
t == null)
                   last: 🕂
ast = new IntList(L.head, null);
                               P does not change!
t.tail = new IntList(L.head, null);
2:48 2019
                                      CS61B: Lecture #4 22
```

ative Non-destructive List Deletion

2:48 2019

but use front-to-back iteration rather than recursion.

CS61B: Lecture #4 21

Destructive Deletion

```
esulting from removing all instances of X from L.

al list may be destroyed. */

dremoveAll(IntList L, int x) {

1)

( null with all x's removed )*/;
lead == x)

( L with all x's removed (L != null) )*/;

// e all x's from L's tail. }*/;

248 2019

C5618: Lecture #4 24
```

Destructive Deletion

```
: Original ..... : after Q = dremoveAll (Q,1)

resulting from removing all instances of X from L.

allist may be destroyed. */;
dremoveAll(IntList L, int x) {
1)
*( null with all x's removed )*/;
lead == x)
*( L with all x's removed (L != null) )*/;

re all x's from L's tail. }*/;
```

Destructive Deletion

```
esulting from removing all instances of X from L.

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

1)

( null with all x's removed )*/;
lead == x)

( L with all x's removed (L != null) )*/;

re all x's from L's tail. }*/;
```

Destructive Deletion

```
: Original ..... : after Q = dremoveAll (Q,1)

resulting from removing all instances of X from L.

tal list may be destroyed. */;
dremoveAll(IntList L, int x) {

1)

(( null with all x's removed )*/;
lead == x)

(( L with all x's removed (L != null) )*/;

re all x's from L's tail. }*/;
```

Destructive Deletion

```
: Original ..... : after Q = dremoveAll (Q,1)

resulting from removing all instances of X from L.

all list may be destroyed. */;

dremoveAll(IntList L, int x) {

1)

lead == x)

removeAll(L.tail, x);

re all x's from L's tail. }*/;

2.48 2019

CS61B: Lecture #4 28
```

Destructive Deletion

2:48 2019

CS61B: Lecture #4 27

Iterative Destructive Deletion

Destructive Deletion

```
: Original ..... : after Q = dremoveAll (Q,1)

esulting from removing all instances of X from L.

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

1)

lead == x)
removeAll(L.tail, x);
dremoveAll(L.tail, x);
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
lt, last;
st = null;
                        P: [-
                                <del>--</del>27<del>1--</del>17<del>1--</del>27<del>1--</del>97\
null) {
xt = L.tail;
                   result: N
 .head) {
                      last: \nabla
 == null)
 = last = L;
 last.tail = L: next:
                                     P = dremoveAll (P, 2)
 null:
lt;
:48 2019
                                             CS61B: Lecture #4 32
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
dremoveAll(IntList L, int x) {
lt, last;
t = null:
                    P: --2 -- 1 -- 2 -- 9N
null) {
xt = L.tail;
                result:
.head) {
                  last:
== null)
= last = L;
                    L: \square
last.tail = L; next:
                               P = dremoveAll (P, 2)
null:
lt;
2:48 2019
                                     CS61B: Lecture #4 31
```

Iterative Destructive Deletion

```
resulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null;
                    P: - 2 - 9 V
null) {
xt = L.tail;
               result: N
.head) {
                 last: 🖯
 == null)
= last = L;
                    냐구
                              P = dremoveAll (P, 2)
 last.tail = L:
 null:
lt;
2:48 2019
                                   CS61B: Lecture #4 34
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null:
null) {
xt = L.tail;
                 result:
.head) {
                   last: 🖯
 == null)
 = last = L;
                  next:口
                                 P = dremoveAll (P. 2)
 last.tail = L:
 null:
lt;
2:48 2019
                                      CS61B: Lecture #4 33
```

Iterative Destructive Deletion

```
resulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null;
null) {
                                       <del>-</del>11 <del>- 2</del> - 9 \
xt = L.tail;
                  result:
.head) {
                    last: 🗔
 == null)
 = last = L;
                                  P = dremoveAll (P, 2)
 last.tail = L:
 null:
lt;
2:48 2019
                                         CS61B: Lecture #4 36
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
velv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null:
                                       <del>-</del>11 <del>-</del>21 <del>-</del>9N
null) {
xt = L.tail;
                  result: \
.head) {
                    last: 🖯
 == null)
 = last = L;
                                   P = dremoveAll (P. 2)
                    next: 🕞
 last.tail = L:
 null:
lt;
2:48 2019
                                          CS61B: Lecture #4 35
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
llt, last;
st = null;
                                           2 - 9
null) {
xt = L.tail;
                result:
 .head) {
                   last: 1
 == null)
 = last = L;
                                P = dremoveAll (P, 2)
 last.tail = L: next:
 null:
lt;
:48 2019
                                      CS61B: Lecture #4 38
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
dremoveAll(IntList L, int x) {
lt, last;
t = null:
                                       <del>-</del>1N 21<del>1 -</del>19N
null) {
xt = L.tail;
                 result:
.head) {
                    last:
 == null)
 = last = L;
                  next: -
                                  P = dremoveAll (P. 2)
 last.tail = L:
 null:
lt;
2.48 2019
                                         CS61B: Lecture #4 37
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null;
null) {
xt = L.tail;
                 result:
 .head) {
                   last: 1
 == null)
 = last = L;
                                 P = dremoveAll (P, 2)
 last.tail = L:
 null:
lt;
2:48 2019
                                       CS61B: Lecture #4 40
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null:
null) {
xt = L.tail;
                 result:
.head) {
 == null)
 = last = L;
                  next: □
                                P = dremoveAll (P, 2)
 last.tail = L:
 null:
lt;
2:48 2019
                                       CS61B: Lecture #4 39
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null;
null) {
xt = L.tail;
                 result:
.head) {
                   last: -
 == null)
 = last = L;
 last.tail = L: next:
                                P = dremoveAll (P, 2)
 null:
lt;
2:48 2019
                                       CS61B: Lecture #4 42
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
velv. */
 dremoveAll(IntList L, int x) {
ilt, last;
st = null:
null) {
xt = L.tail;
                 result: [
.head) {
 == null)
 = last = L;
                                 P = dremoveAll (P, 2)
                  next: 🖯
 last.tail = L:
 null:
lt;
2:48 2019
                                       CS61B: Lecture #4 41
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
relv. */
 dremoveAll(IntList L, int x) {
llt, last;
st = null;
null) {
xt = L.tail;
                 result:
 .head) {
                   last: -
 == null)
 = last = L;
                     L: 🔽
 last.tail = L; next:
                                 P = dremoveAll (P, 2)
 null:
Lt;
:48 2019
                                       CS61B: Lecture #4 44
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
rely. */
 dremoveAll(IntList L, int x) {
lt, last;
t = null;
null) {
xt = L.tail;
                 result:
.head) {
 == null)
 = last = L;
                     L:N
last.tail = L; next:∑
                                P = dremoveAll (P, 2)
 null:
lt;
2:48 2019
                                       CS61B: Lecture #4 43
```

Relationship to Recursion

to see this is to consider an equivalent recursive pro-

```
g Invariant, produce a situation where Inveriant and condition is false. */
{
riant assumed true here.
dition) {
body
Invariant must be true here.
p()
Invariant true here and condition false.
```

variant is the precondition of the function loop.

ntains the invariant while making the condition false.

range that our actual goal is implied by this post-condition.

2:48 2019 C561B: Lecture #4 46

e: How to Write a Loop (in Theory)

description of how things look on *any arbitrary itera*pop.

tion is known as a *loop invariant*, because it is always tart of each iteration.

ly then must

m any situation consistent with the invariant;

press in such a way as to make the invariant true again.

riant must be true here

condition) { // condition must not have side-effects.
nvariant will necessarily be true here.)

bodv

variant must again be true here

riant true and condition false.

p gets the desired answer whenever *Invariant* is true talse, our job is done!

P:48 2019 CS61B: Lecture #4 45

nple: Loop Invariant for dremoveAll

```
ulting from removing all X's from L
 . */
 emoveAll(IntList L, int x) {
 last:
 null;
null) {
                   result:
= L.tail;
ead) {
 null)
last = L;
                                     P = dremoveAll (P, 2)
ast.tail = L;
                   ** Invariant:
h11.
                    • result points to the list of items in the
                      final result except for those from L on-
                      ward.
                    • L points to an unchanged tail of the
                      original list of items in L.
                    • last points to the last item in result
                      or is null if result is null.
2:48 2019
                                            CS61B: Lecture #4 47
```