Public Service Announcement I

working with kids? Do you like making a positive
r youth? Do you like meeting amazing and congenial
OASES now!
anization of 150 mentors, we tutor elementary school
nd. This is an fantastic opportunity serve as an im-
model for under-resourced children. From playing
lping them with homework, every moment makes a
You will also meet new and like-minded people eager
youth! Also, you can earn either an Education field
or an Asian American Studies unit!
nfo-sessions from Tuesday, Sept 5th to Friday, Sept
-6:30 PM at the Free Speech Movement Cafe)
s? Contact leadcoords.oases@gmail.com. We're also
www.facebook.com/OasesAtUcBerkeley/."

Public Service Announcement II

join the Berkeley Political Review! Berkeley Political
Berkeley's only non-partisan undergraduate political
lding our last info session, next Tuesday September
cation TBD-see FB event for more details). BPR is
riters, business and marketing professionals, tech
designers—come find your place in the BPR family!
are due September 7th. Apply online at
berkeley.edu/apply/."

Recreation

very acute angle $\alpha > 0$,

$$\tan\alpha + \cot\alpha \geq 2$$

ecture #5: Simple Pointer Manipulation

pointer hacking.

labs and homework: We'll be lenient about accepting
rk and labs for the first few. Just get it done: part
is getting to understand the tools involved. We will *not*
issions by email.

Destructive Incrementing

utions may modify objects in the original list to save

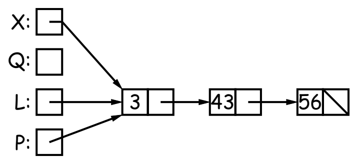
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

o more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



Destructive Incrementing

utions may modify objects in the original list to save

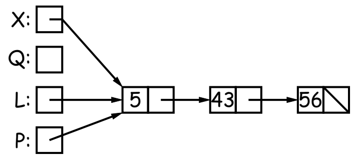
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

o more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



Destructive Incrementing

utions may modify objects in the original list to save

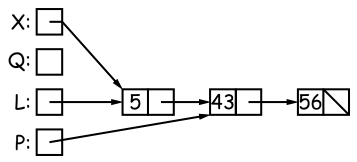
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

o more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



Destructive Incrementing

utions may modify objects in the original list to save

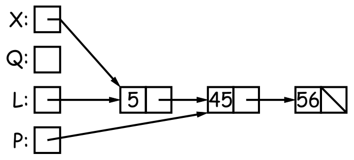
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

 more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



Destructive Incrementing

utions may modify objects in the original list to save

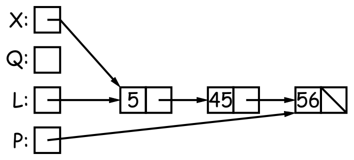
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

 more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



Destructive Incrementing

utions may modify objects in the original list to save

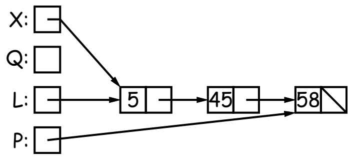
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

 more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



Destructive Incrementing

utions may modify objects in the original list to save

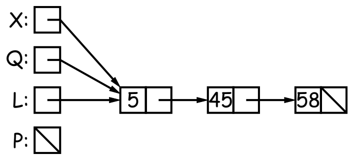
```
y add N to L's items. */
incrList(IntList P, int n) {

crList(P.tail, n);

y add N to L's items. */
incrList(IntList L, int n)

 more than count!
 = L; p != null; p = p.tail)
```

```
X = IntList.list(3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList(X, 2);
```



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
ictively. */
 removeAll(IntList L, int x) {
l1)
*( null with all x's removed )*/;
ead == x)
*( L with all x's removed (L!=null, L.head==x) )*/;

*( L with all x's removed (L!=null, L.head!=x) )*/;
```

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
ictively. */
 removeAll(IntList L, int x) {
l1)
ll;
ead == x)
*( L with all x's removed (L!=null, L.head==x) )*/;

*( L with all x's removed (L!=null, L.head!=x) )*/;
```

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
ictively. */
: removeAll(IntList L, int x) {
l)
ll;
head == x)
moveAll(L.tail, x);

w IntList(L.head, removeAll(L.tail, x));
```

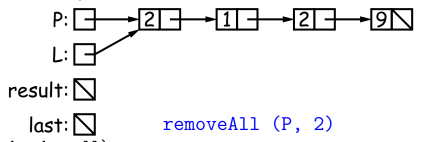
ative Non-destructive List Deletion

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

```
ulting from removing all instances
ion-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);
```



removeAll (P, 2)

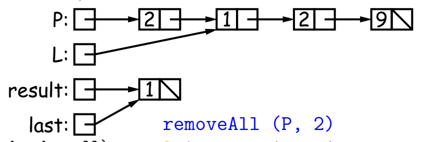
ative Non-destructive List Deletion

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

```
ulting from removing all instances
ion-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);
```



removeAll (P, 2)

P does *not* change!

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
ictively. */
: removeAll(IntList L, int x) {
l)
ll;
head == x)
moveAll(L.tail, x);

*( L with all x's removed (L!=null, L.head!=x) )*/;
```

ative Non-destructive List Deletion

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

```
ulting from removing all instances
ion-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);
```

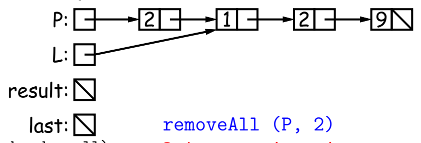
ative Non-destructive List Deletion

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

```
ulting from removing all instances
ion-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);
```



removeAll (P, 2)

P does *not* change!

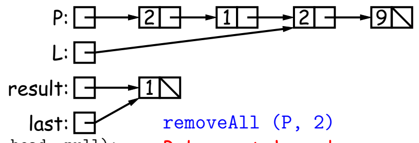
ative Non-destructive List Deletion

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

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

t == null)
ast = new IntList(L.head, null);

t.tail = new IntList(L.head, null);
```



removeAll (P, 2)

P does not change!

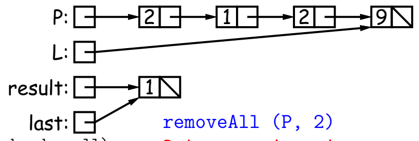
ative Non-destructive List Deletion

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

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

t == null)
ast = new IntList(L.head, null);

t.tail = new IntList(L.head, null);
```



removeAll (P, 2)

P does not change!

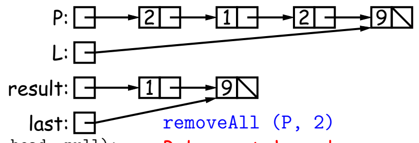
ative Non-destructive List Deletion

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

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

t == null)
ast = new IntList(L.head, null);

t.tail = new IntList(L.head, null);
```



removeAll (P, 2)

P does not change!

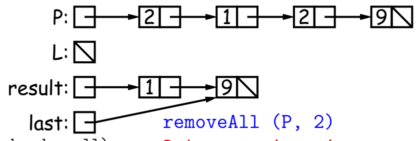
ative Non-destructive List Deletion

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

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

t == null)
ast = new IntList(L.head, null);

t.tail = new IntList(L.head, null);
```

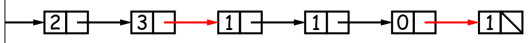


removeAll (P, 2)

P does not change!

Destructive Deletion

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

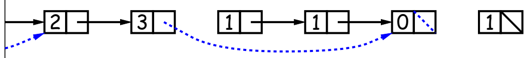


```
resulting from removing all instances of X from L.
al list may be destroyed. */
: dremoveAll(IntList L, int x) {
ll)
*( null with all x's removed )*/;
head == x)
*( L with all x's removed (L != null) )*/;

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

Destructive Deletion

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

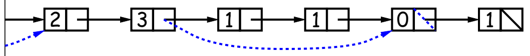


```
resulting from removing all instances of X from L.
al list may be destroyed. */
: dremoveAll(IntList L, int x) {
ll)
*( null with all x's removed )*/;
head == x)
*( L with all x's removed (L != null) )*/;

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

Destructive Deletion

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

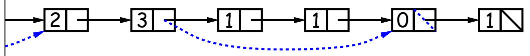


```
esulting from removing all instances of X from L.
al list may be destroyed. */
 dremoveAll(IntList L, int x) {
l1)
( null with all x's removed )*/;
ead == x)
( L with all x's removed (L != null) )*/;

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

Destructive Deletion

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

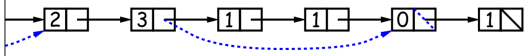


```
esulting from removing all instances of X from L.
al list may be destroyed. */
 dremoveAll(IntList L, int x) {
l1)
( null with all x's removed )*/;
ead == x)
( L with all x's removed (L != null) )*/;

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

Destructive Deletion

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

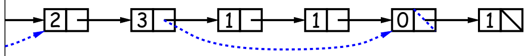


```
esulting from removing all instances of X from L.
al list may be destroyed. */
 dremoveAll(IntList L, int x) {
l1)
ll;
ead == x)
( L with all x's removed (L != null) )*/;

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

Destructive Deletion

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



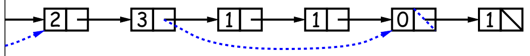
```
esulting from removing all instances of X from L.
al list may be destroyed. */
 dremoveAll(IntList L, int x) {
l1)

ead == x)
emoveAll(L.tail, x);

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

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) {
l1)

ead == x)
emoveAll(L.tail, x);

 dremoveAll(L.tail, x);
```

Iterative Destructive Deletion

```
esulting from removing all X's from L
ely. */
 dremoveAll(IntList L, int x) {
lt, last;
t = null;
 null) {
xt = L.tail;
.head) {
 == null)
 = last = L;

 last.tail = L;
 null;



t;
```

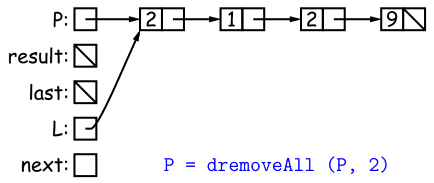
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
: dremoveAll(IntList L, int x) {
ilt, last;
st = null;
 null) {
ext = L.tail;
.head) {
: == null)
: = last = L;

: last.tail = L;
: null;




.t;
```



P = dremoveAll (P, 2)

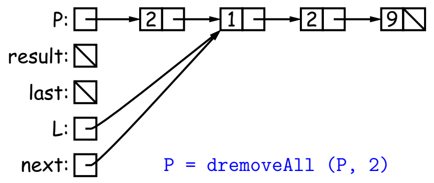
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
: dremoveAll(IntList L, int x) {
ilt, last;
st = null;
 null) {
ext = L.tail;
.head) {
: == null)
: = last = L;

: last.tail = L;
: null;




.t;
```



P = dremoveAll (P, 2)

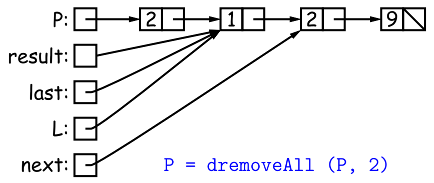
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
: dremoveAll(IntList L, int x) {
ilt, last;
st = null;
 null) {
ext = L.tail;
.head) {
: == null)
: = last = L;

: last.tail = L;
: null;




.t;
```



P = dremoveAll (P, 2)

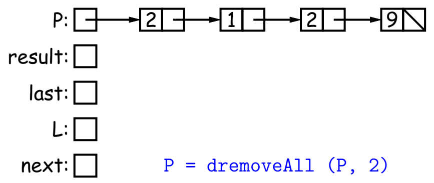
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
: dremoveAll(IntList L, int x) {
ilt, last;
st = null;
 null) {
ext = L.tail;
.head) {
: == null)
: = last = L;

: last.tail = L;
: null;




.t;
```



P = dremoveAll (P, 2)

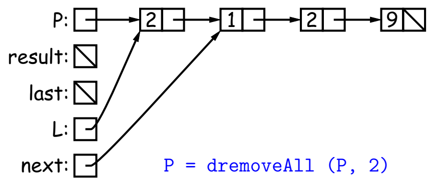
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
: dremoveAll(IntList L, int x) {
ilt, last;
st = null;
 null) {
ext = L.tail;
.head) {
: == null)
: = last = L;

: last.tail = L;
: null;




.t;
```



P = dremoveAll (P, 2)

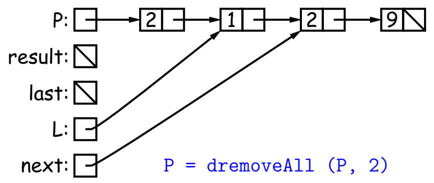
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
: dremoveAll(IntList L, int x) {
ilt, last;
st = null;
 null) {
ext = L.tail;
.head) {
: == null)
: = last = L;

: last.tail = L;
: null;




.t;
```



P = dremoveAll (P, 2)

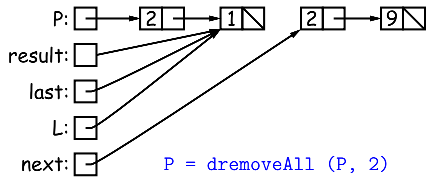
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
; dremoveAll(IntList L, int x) {
ılt, last;
st = null;
 null) {
ext = L.tail;
.head) {
; == null)
; = last = L;

= last.tail = L;
= null;



t;
```



P = dremoveAll (P, 2)

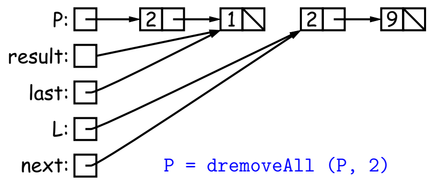
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
; dremoveAll(IntList L, int x) {
ılt, last;
st = null;
 null) {
ext = L.tail;
.head) {
; == null)
; = last = L;

= last.tail = L;
= null;



t;
```



P = dremoveAll (P, 2)

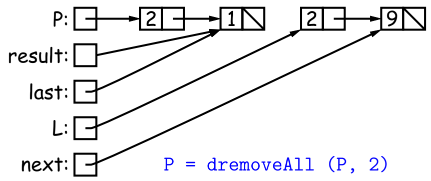
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
; dremoveAll(IntList L, int x) {
ılt, last;
st = null;
 null) {
ext = L.tail;
.head) {
; == null)
; = last = L;

= last.tail = L;
= null;



t;
```



P = dremoveAll (P, 2)

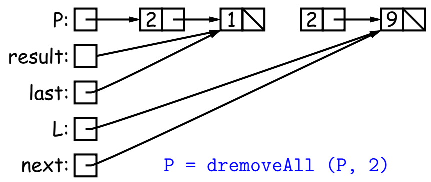
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
; dremoveAll(IntList L, int x) {
ılt, last;
st = null;
 null) {
ext = L.tail;
.head) {
; == null)
; = last = L;

= last.tail = L;
= null;



t;
```



P = dremoveAll (P, 2)

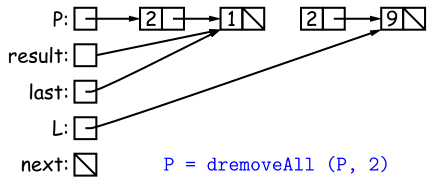
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
; dremoveAll(IntList L, int x) {
ılt, last;
st = null;
 null) {
ext = L.tail;
.head) {
; == null)
; = last = L;

= last.tail = L;
= null;



t;
```



P = dremoveAll (P, 2)

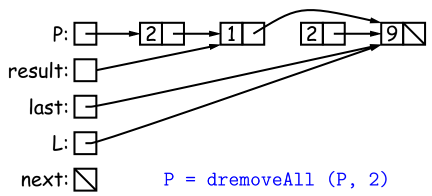
Iterative Destructive Deletion

```
resulting from removing all X's from L
vely. */
; dremoveAll(IntList L, int x) {
ılt, last;
st = null;
 null) {
ext = L.tail;
.head) {
; == null)
; = last = L;

= last.tail = L;
= null;



t;
```



P = dremoveAll (P, 2)

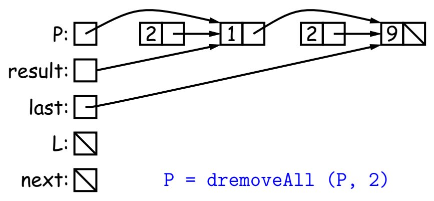
Iterative Destructive Deletion

```
esulting from removing all X's from L
ely. */
 dremoveAll(IntList L, int x) {
lt, last;
t = null;
 null) {
xt = L.tail;
.head) {
 == null)
 = last = L;

 last.tail = L;
 null;



t;
```



P = dremoveAll (P, 2)

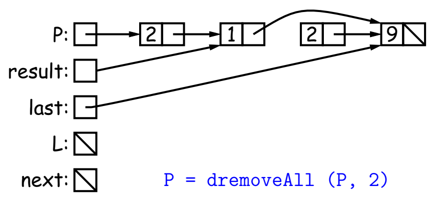
Iterative Destructive Deletion

```
esulting from removing all X's from L
ely. */
 dremoveAll(IntList L, int x) {
lt, last;
t = null;
 null) {
xt = L.tail;
.head) {
 == null)
 = last = L;

 last.tail = L;
 null;



t;
```



P = dremoveAll (P, 2)