

## Public Service Announcement II

Join the Berkeley Political Review! Berkeley Political Review is Berkeley's only non-partisan undergraduate political organization (following our last info session, next Tuesday September 24th 7-9pm, see Facebook event for more details). BPR is currently seeking writers, business and marketing professionals, tech designers—come find your place in the BPR family! Applications are due September 7th. Apply online at [berkeley.edu/apply/](http://berkeley.edu/apply/).

## Recreation

Every acute angle  $\alpha > 0$ ,

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

58:48 2017

CS61B: Lecture #5 2

## Destructive Incrementing

Operations may modify objects in the original list to save

```
function incrList(IntList P, int n) {
  X = IntList.list(3, 43, 56);
  /* IntList.list from HW #1 */
  Q = dincrList(X, 2);

  incrList(P.tail, n);
  X: [ ]
  Q: [ ]
  L: [ ]
  P: [ ]
  3 [ ] → 43 [ ] → 56 [ ]
}

/* add N to L's items. */
incrList(IntList L, int n) {
  if (L == null) return;
  if (L.tail == null) {
    L.data = L.data + n;
  } else {
    incrList(L.tail, n);
  }
}
```

58:48 2017

CS61B: Lecture #5 4

## Destructive Incrementing

Operations may modify objects in the original list to save

```
function incrList(IntList P, int n) {
  X = IntList.list(3, 43, 56);
  /* IntList.list from HW #1 */
  Q = dincrList(X, 2);

  incrList(P.tail, n);
  X: [ ]
  Q: [ ]
  L: [ ]
  P: [ ]
  5 [ ] → 43 [ ] → 56 [ ]
}

/* add N to L's items. */
incrList(IntList L, int n) {
  if (L == null) return;
  if (L.tail == null) {
    L.data = L.data + n;
  } else {
    incrList(L.tail, n);
  }
}
```

58:48 2017

CS61B: Lecture #5 6

## Public Service Announcement I

Working with kids? Do you like making a positive impact on youth? Do you like meeting amazing and congenial people? OASES now!

Organization of 150 mentors, we tutor elementary school children. This is an fantastic opportunity serve as an informal model for under-resourced children. From playing with them with homework, every moment makes a difference. You will also meet new and like-minded people eager to help youth! Also, you can earn either an Education field or an Asian American Studies unit!

Info-sessions from Tuesday, Sept 5th to Friday, Sept 8th 6-8:30 PM at the Free Speech Movement Cafe

Interested? Contact [leadcoords.oases@gmail.com](mailto:leadcoords.oases@gmail.com). We're also on Facebook: [www.facebook.com/OasesAtUcBerkeley/](http://www.facebook.com/OasesAtUcBerkeley/).

58:48 2017

CS61B: Lecture #5 1

## Lecture #5: Simple Pointer Manipulation

Simple pointer hacking.

**Labs and homework:** We'll be lenient about accepting late work and labs for the first few. Just get it done: part of the goal is getting to understand the tools involved. We will *not* accept late assignments by email.

58:48 2017

CS61B: Lecture #5 3

## Destructive Incrementing

Operations may modify objects in the original list to save

```
function incrList(IntList P, int n) {
  X = IntList.list(3, 43, 56);
  /* IntList.list from HW #1 */
  Q = dincrList(X, 2);

  incrList(P.tail, n);
  X: [ ]
  Q: [ ]
  L: [ ]
  P: [ ]
  5 [ ] → 43 [ ] → 56 [ ]
}

/* add N to L's items. */
incrList(IntList L, int n) {
  if (L == null) return;
  if (L.tail == null) {
    L.data = L.data + n;
  } else {
    incrList(L.tail, n);
  }
}
```

58:48 2017

CS61B: Lecture #5 5

## Destructive Incrementing

Mutations may modify objects in the original list to save

```
by 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);

  incrList(P.tail, n);
}

by add N to L's items. */
incrList(IntList L, int n)
  p more than count!
  = L; p != null; p = p.tail)
```

58:48 2017

CS61B: Lecture #5 8

## Destructive Incrementing

Mutations may modify objects in the original list to save

```
by 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);

  incrList(P.tail, n);
}

by add N to L's items. */
incrList(IntList L, int n)
  p more than count!
  = L; p != null; p = p.tail)
```

58:48 2017

CS61B: Lecture #5 10

## 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
destructively. */
removeAll(IntList L, int x) {
  l
  all;
  head == x)
  *( L with all x's removed (L!=null, L.head==x) )*/;
  *( L with all x's removed (L!=null, L.head!=x) )*/;
```

58:48 2017

CS61B: Lecture #5 12

## Destructive Incrementing

Mutations may modify objects in the original list to save

```
by 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);

  incrList(P.tail, n);
}

by add N to L's items. */
incrList(IntList L, int n)
  p more than count!
  = L; p != null; p = p.tail)
```

58:48 2017

CS61B: Lecture #5 7

## Destructive Incrementing

Mutations may modify objects in the original list to save

```
by 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);

  incrList(P.tail, n);
}

by add N to L's items. */
incrList(IntList L, int n)
  p more than count!
  = L; p != null; p = p.tail)
```

58:48 2017

CS61B: Lecture #5 9

## 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
destructively. */
removeAll(IntList L, int x) {
  l
  head == 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) )*/;
```

58:48 2017

CS61B: Lecture #5 11

### 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  
actively. \*/

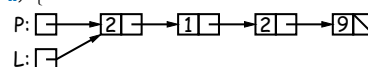

```
removeAll(IntList L, int x) {  
  L  
  null;  
  head == x)  
  removeAll(L.tail, x);  
  new IntList(L.head, removeAll(L.tail, x));  
}
```

### Active Non-destructive List Deletion

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

alternating from removing all instances

non-destructively. \*/

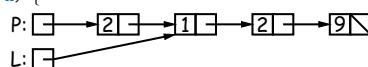

```
removeAll(IntList L, int x) {  
  P, last;  
  = null;  
  while (L = L.tail) {  
    head)  
    result:   
    t == null) last:  removeAll (P, 2)  
    last = new IntList(L.head, null);  
    t.tail = new IntList(L.head, null);  
  }  
}
```

### Active Non-destructive List Deletion

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

alternating from removing all instances

non-destructively. \*/

```
removeAll(IntList L, int x) {  
  P, last;  
  = null;  
  while (L = L.tail) {  
    head)  
    result:   
    t == null) last:  removeAll (P, 2)  
    last = new IntList(L.head, null); P does not change!  
    t.tail = new IntList(L.head, null);  
  }  
}
```

### 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  
actively. \*/

```
removeAll(IntList L, int x) {  
  L  
  null;  
  head == x)  
  removeAll(L.tail, x);  
  new IntList(L.head, removeAll(L.tail, x));  
}
```

### Active Non-destructive List Deletion

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

alternating from removing all instances

non-destructively. \*/

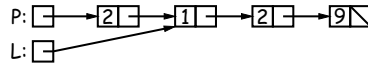

```
removeAll(IntList L, int x) {  
  P, last;  
  = null;  
  while (L = L.tail) {  
    head)  
    t == null)  
    last = new IntList(L.head, null);  
    t.tail = new IntList(L.head, null);  
  }  
}
```

### Active Non-destructive List Deletion

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

alternating from removing all instances

non-destructively. \*/

```
removeAll(IntList L, int x) {  
  P, last;  
  = null;  
  while (L = L.tail) {  
    head)  
    result:   
    t == null) last:  removeAll (P, 2)  
    last = new IntList(L.head, null); P does not change!  
    t.tail = new IntList(L.head, null);  
  }  
}
```

## Active Non-destructive List Deletion

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

Resulting from removing all instances

non-destructively. \*/

```
removeAll(IntList L, int x) {
    last;
    = null;
    all; L = L.tail) {
        head)
        result: [ ] -> [1] -> [ ]
        last: [ ] -> [2] -> [1] -> [2] -> [9]
        removeAll(P, 2)
        P does not change!
    }
    L.tail = new IntList(L.head, null);
}
```

58:48 2017

CS61B: Lecture #5 20

## Active Non-destructive List Deletion

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

Resulting from removing all instances

non-destructively. \*/

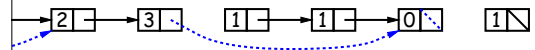
```
removeAll(IntList L, int x) {
    last;
    = null;
    all; L = L.tail) {
        head)
        result: [ ] -> [1] -> [9]
        last: [ ] -> [2] -> [1] -> [2] -> [9]
        removeAll(P, 2)
        P does not change!
    }
    L.tail = new IntList(L.head, null);
}
```

58:48 2017

CS61B: Lecture #5 22

## Destructive Deletion

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



Resulting from removing all instances of X from L.

tail list may be destroyed. \*/

```
dremoveAll(IntList L, int x) {
    l)
    *( null with all x's removed )*/;
    head == x)
    *( L with all x's removed (L != null) )*/;
    re all x's from L's tail. }*/;
}
```

58:48 2017

CS61B: Lecture #5 24

## Active Non-destructive List Deletion

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

Resulting from removing all instances

non-destructively. \*/

```
removeAll(IntList L, int x) {
    last;
    = null;
    all; L = L.tail) {
        head)
        result: [ ] -> [1] -> [ ]
        last: [ ] -> [2] -> [1] -> [2] -> [9]
        removeAll(P, 2)
        P does not change!
    }
    L.tail = new IntList(L.head, null);
}
```

58:48 2017

CS61B: Lecture #5 19

## Active Non-destructive List Deletion

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

Resulting from removing all instances

non-destructively. \*/

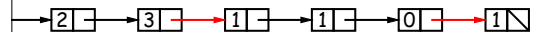
```
removeAll(IntList L, int x) {
    last;
    = null;
    all; L = L.tail) {
        head)
        result: [ ] -> [1] -> [9]
        last: [ ] -> [2] -> [1] -> [2] -> [9]
        removeAll(P, 2)
        P does not change!
    }
    L.tail = new IntList(L.head, null);
}
```

58:48 2017

CS61B: Lecture #5 21

## Destructive Deletion

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



Resulting from removing all instances of X from L.

tail list may be destroyed. \*/

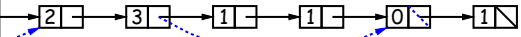
```
dremoveAll(IntList L, int x) {
    l)
    *( null with all x's removed )*/;
    head == x)
    *( L with all x's removed (L != null) )*/;
    re all x's from L's tail. }*/;
}
```

58:48 2017

CS61B: Lecture #5 23

## Destructive Deletion

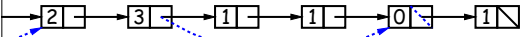
: Original      ..... : after Q = dremoveAll (Q,1)



```
resulting from removing all instances of X from L.
tail list may be destroyed. */
; dremoveAll(IntList L, int x) {
  l)
  *( 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)



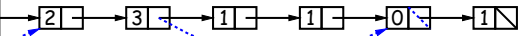
```
resulting from removing all instances of X from L.
tail list may be destroyed. */
; dremoveAll(IntList L, int x) {
  l)
  lead == x)
  removeAll(L.tail, x);
  re all x's from L's tail. }*/;
```

## Iterative Destructive Deletion

```
resulting from removing all X's from L
rely. */
; dremoveAll(IntList L, int x) {
  l, last;
  st = null;
  null) {
  xt = L.tail;
  l.head) {
  ; == null)
  ; = last = L;
  = last.tail = L;
  = null;
  t;
  t;
```

## Destructive Deletion

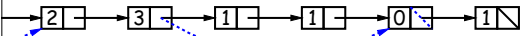
: Original      ..... : after Q = dremoveAll (Q,1)



```
resulting from removing all instances of X from L.
tail list may be destroyed. */
; dremoveAll(IntList L, int x) {
  l)
  *( 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

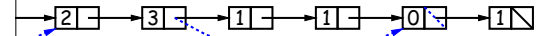
: Original      ..... : after Q = dremoveAll (Q,1)



```
resulting from removing all instances of X from L.
tail list may be destroyed. */
; dremoveAll(IntList L, int x) {
  l)
  all;
  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.
tail list may be destroyed. */
; dremoveAll(IntList L, int x) {
  l)
  lead == x)
  removeAll(L.tail, x);
  dremoveAll(L.tail, x);
  }
```

### Iterative Destructive Deletion

resulting from removing all X's from L

```

1 // rely. */
2 dremoveAll(IntList L, int x) {
3     lt, last;
4     st = null;
5     null) {
6         xt = L.tail;
7         L.head) {
8             == null)
9             = last = L;
10
11     = last.tail = L;
12     = null;
13
14
15 t;

```

P = dremoveAll (P, 2)

### Iterative Destructive Deletion

resulting from removing all X's from L

```

1 // rely. */
2 dremoveAll(IntList L, int x) {
3     lt, last;
4     st = null;
5     null) {
6         xt = L.tail;
7         L.head) {
8             == null)
9             = last = L;
10
11     = last.tail = L;
12     = null;
13
14
15 t;

```

P = dremoveAll (P, 2)

### Iterative Destructive Deletion

resulting from removing all X's from L

```

1 // rely. */
2 dremoveAll(IntList L, int x) {
3     lt, last;
4     st = null;
5     null) {
6         xt = L.tail;
7         L.head) {
8             == null)
9             = last = L;
10
11     = last.tail = L;
12     = null;
13
14
15 t;

```

P = dremoveAll (P, 2)

### Iterative Destructive Deletion

resulting from removing all X's from L

```

1 // rely. */
2 dremoveAll(IntList L, int x) {
3     lt, last;
4     st = null;
5     null) {
6         xt = L.tail;
7         L.head) {
8             == null)
9             = last = L;
10
11     = last.tail = L;
12     = null;
13
14
15 t;

```

P = dremoveAll (P, 2)

### Iterative Destructive Deletion

resulting from removing all X's from L

```

1 // rely. */
2 dremoveAll(IntList L, int x) {
3     lt, last;
4     st = null;
5     null) {
6         xt = L.tail;
7         L.head) {
8             == null)
9             = last = L;
10
11     = last.tail = L;
12     = null;
13
14
15 t;

```

P = dremoveAll (P, 2)

### Iterative Destructive Deletion

resulting from removing all X's from L

```

1 // rely. */
2 dremoveAll(IntList L, int x) {
3     lt, last;
4     st = null;
5     null) {
6         xt = L.tail;
7         L.head) {
8             == null)
9             = last = L;
10
11     = last.tail = L;
12     = null;
13
14
15 t;

```

P = dremoveAll (P, 2)

### Iterative Destructive Deletion

resulting from removing all X's from L

```

rely. */
; dremoveAll(IntList L, int x) {
lt, last;
st = null;
null) {
xt = L.tail;
..head) {
; == null)
; = last = L;
= last.tail = L;
= null;

```

P = dremoveAll(P, 2)

t;

### Iterative Destructive Deletion

resulting from removing all X's from L

```

rely. */
; dremoveAll(IntList L, int x) {
lt, last;
st = null;
null) {
xt = L.tail;
..head) {
; == null)
; = last = L;
= last.tail = L;
= null;

```

P = dremoveAll(P, 2)

t;

### Iterative Destructive Deletion

resulting from removing all X's from L

```

rely. */
; dremoveAll(IntList L, int x) {
lt, last;
st = null;
null) {
xt = L.tail;
..head) {
; == null)
; = last = L;
= last.tail = L;
= null;

```

P = dremoveAll(P, 2)

t;

### Iterative Destructive Deletion

resulting from removing all X's from L

```

rely. */
; dremoveAll(IntList L, int x) {
lt, last;
st = null;
null) {
xt = L.tail;
..head) {
; == null)
; = last = L;
= last.tail = L;
= null;

```

P = dremoveAll(P, 2)

t;

### Iterative Destructive Deletion

resulting from removing all X's from L

```

rely. */
; dremoveAll(IntList L, int x) {
lt, last;
st = null;
null) {
xt = L.tail;
..head) {
; == null)
; = last = L;
= last.tail = L;
= null;

```

P = dremoveAll(P, 2)

t;

### Iterative Destructive Deletion

resulting from removing all X's from L

```

rely. */
; dremoveAll(IntList L, int x) {
lt, last;
st = null;
null) {
xt = L.tail;
..head) {
; == null)
; = last = L;
= last.tail = L;
= null;

```

P = dremoveAll(P, 2)

t;

## Iterative Destructive Deletion

resulting from removing all X's from L

```
rely. */
; dremoveAll(IntList L, int x) {
  lt, last;
  st = null;
  null) {
  xt = L.tail;
  .head) {
  == null)
  = last = L;
  = last.tail = L;
  = null;

  t;
```

The diagram illustrates the state of pointers during the execution of the `dremoveAll` function. A pointer `P` points to the first node of a linked list containing nodes with values 2, 1, 2, and 9. The `result` and `last` pointers also point to the first node. The `L` and `next` pointers point to null. The code snippet shows the function signature and the beginning of the implementation, including the recursive call `P = dremoveAll (P, 2)`.

## Iterative Destructive Deletion

resulting from removing all X's from L

```
rely. */
; dremoveAll(IntList L, int x) {
  lt, last;
  st = null;
  null) {
  xt = L.tail;
  .head) {
  == null)
  = last = L;
  = last.tail = L;
  = null;

  t;
```

The diagram illustrates the state of pointers during the execution of the `dremoveAll` function. A pointer `P` points to the first node of a linked list containing nodes with values 2, 1, 2, and 9. The `result` and `last` pointers also point to the first node. The `L` and `next` pointers point to null. The code snippet shows the function signature and the beginning of the implementation, including the recursive call `P = dremoveAll (P, 2)`.