Sentinels

Dr. Mattox Beckman

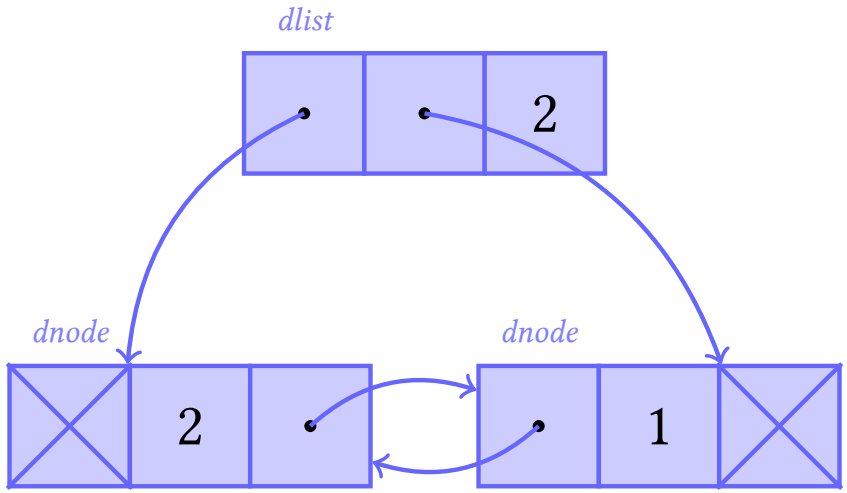
Illinois Institute of Technology
Department of Computer Science

Objectives

- Understand how to form a sentinel.
- Show how to use a sentinel to replace `null`.
- Show how to use a sentinel to remove edge cases.

Our Doubly Linked List So Far

```
1       (struct dlist (front back size) #:mutable)
2       (struct dnode (prev data next) #:mutable)
3       (define (make-dlist) (dlist null null 0))
```



Ordered Insert: Helper Functions

- To insert, we need to find the proper insertion point first.
- Then we can link the node into the doubly linked list.

```
(define (is-empty? xx)
  (null? (dlist-front xx)))

(define (find-greater dns elt)
  (cond ((null? dns) null)
        ((< elt (dnode-data dns)) dns)
        (else (find-greater (dnode-next dns) elt))
        ))
```

Insert Front: Outline

```
1  (define (ordered-insert xx elt)
2    (inc-dlist-size! xx)
3    (let ((node (dnode null elt null))
4          (target (find-greater (dlist-front xx) elt)))
5      (cond ((is-empty? xx)  ; nothing in the list
6             ; ...
7            ((null? target)  ; last element
8             ; ...
9            ((null? (dnode-prev target))   ; first element
10            ; ...
11           (else (begin
12            ; ...
13           )))
```

Insert Front: Empty Case

```
1  (define (ordered-insert xx elt)
2    (inc-dlist-size! xx)
3    (let ((node (dnode null elt null))
4          (target (find-greater (dlist-front xx) elt)))
5      (cond ((is-empty? xx)  ; nothing in the list
6             (begin (set-dlist-front! xx node)
7                    (set-dlist-back! xx node)) )
8            ((null? target)  ; last element
9             ; ...
10           ((null? (dnode-prev target))   ; first element
11            ; ...
12           (else (begin
13            ; ...
14           )))
```

Insert Front: Last element

```
1  (define (ordered-insert xx elt)
2    (inc-dlist-size! xx)
3    (let ((node (dnode null elt null))
4          (target (find-greater (dlist-front xx) elt)))
5      (cond ((is-empty? xx)  ; nothing in the list
6              ; ...
7             ((null? target)  ; last element
8              (begin (set-dnode-prev! node (dlist-back xx))
9                     (set-dnode-next! (dnode-prev node) node)
10                    (set-dlist-back! xx node)))
11            ((null? (dnode-prev target))   ; first element
12             ; ...
13            (else (begin
14             ; ...
15            )))
```

Insert Front: First Element

```
1  (define (ordered-insert xx elt)
2    (inc-dlist-size! xx)
3    (let ((node (dnode null elt null))
4          (target (find-greater (dlist-front xx) elt)))
5      (cond ((is-empty? xx)  ; nothing in the list
6             ; ...
7            ((null? target)  ; last element
8             ; ...
9            ((null? (dnode-prev target))   ; first element
10            (begin (set-dnode-next! node target)
11                   (set-dnode-prev! target node)
12                   (set-dlist-front! xx node)))
13           (else (begin
14            ; ...
15           )))
```

Insert Front: Middle Element

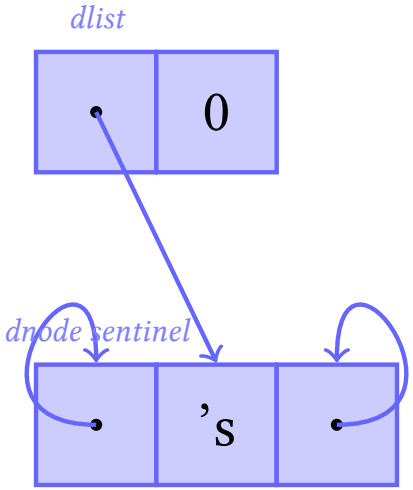
```
1  (define (ordered-insert xx elt)
2    (inc-dlist-size! xx)
3    (let ((node (dnode null elt null))
4          (target (find-greater (dlist-front xx) elt)))
5      (cond ((is-empty? xx)  ; nothing in the list
6             ; ...
7            ((null? target)  ; last element
8             ; ...
9            ((null? (dnode-prev target))   ; first element
10            ; ...
11           (else (begin
12                   (set-dnode-next! node target)
13                   (set-dnode-prev! node (dnode-prev target))
14                   (set-dnode-next! (dnode-prev node) node)
15                   (set-dnode-prev! (dnode-next node) node)))
16           )))
```

Old Insert Front

```
(define (insert-front xx elt)
  (let ((node (dnode null elt (dlist-front xx))))
    (cond ((null? (dlist-front xx))
           (begin (set-dlist-front! xx node)
                  (set-dlist-back! xx node)
                  (inc-dlist-size! xx)))
          (else (begin (set-dnode-prev! (dlist-front xx) node)
                       (set-dlist-front! xx node)
                       (inc-dlist-size! xx))))
    ))
```

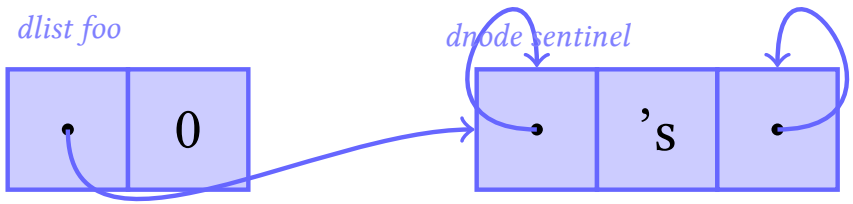
Introducting Sentinels!

```
1 (struct dlist (sentinel size) #:mutable)
2 (struct dnode (prev data next) #:mutable)
3 (define (make-dlist)
4   (let ((sentinel (dnode null 'sentinel null)))
5     (begin
6       (set-dnode-prev! sentinel sentinel)
7       (set-dnode-next! sentinel sentinel)
8       (dlist sentinel 0))))
```



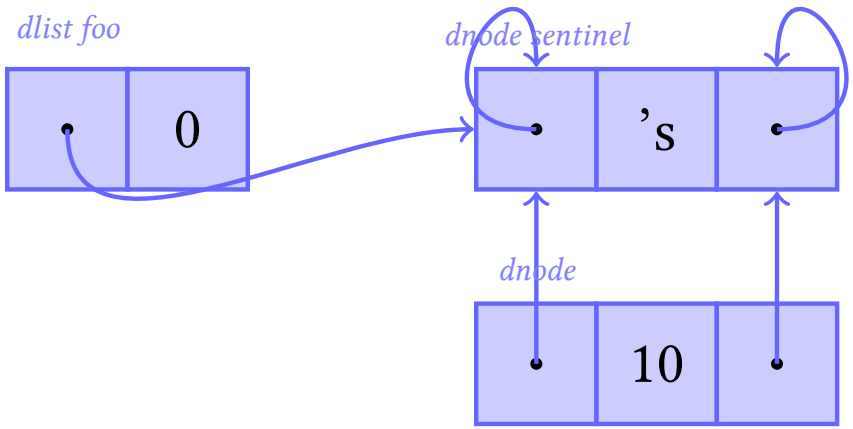
Insert Front

```
1 (define (insert-front xx elt)
2   (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)))))
3     (begin
4       (set-dnode-prev! (dnode-next node) node)
5       (set-dnode-next! (dlist-sentinel xx) node)
6       (inc-dlist-size! dlist))))
7 (define foo (make-dlist))
8 (insert foo 10)
```



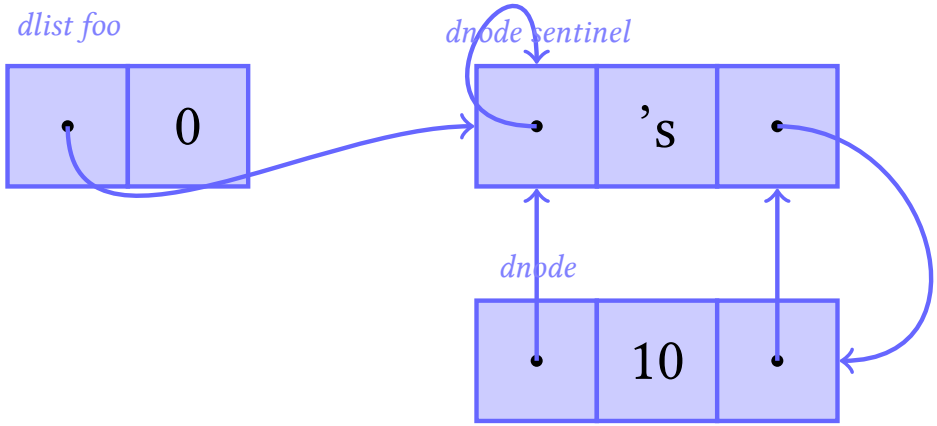
Insert Front

```
1 (define (insert-front xx elt)
2   (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)
3     (begin
4       (set-dnode-prev! (dnode-next node) node)
5       (set-dnode-next! (dlist-sentinel xx) node)
6       (inc-dlist-size! dlist))))
7 (define foo (make-dlist))
8 (insert foo 10)
```



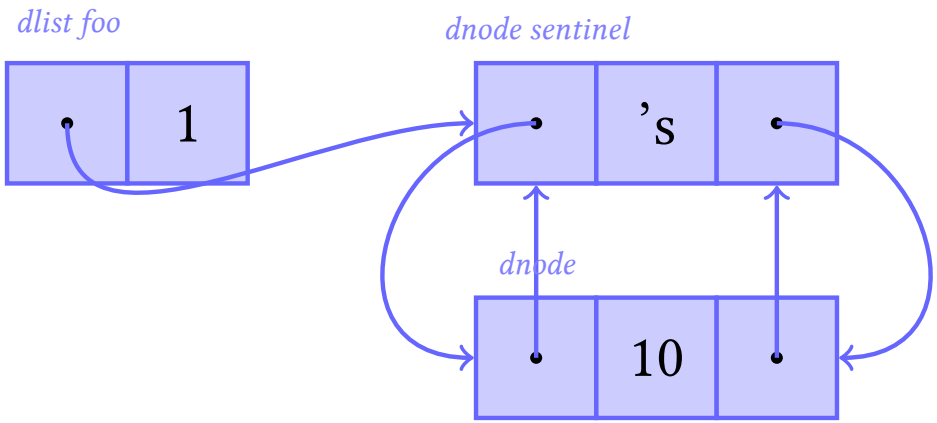
Insert Front

```
1 (define (insert-front xx elt)
2   (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)))))
3     (begin
4       (set-dnode-prev! (dnode-next node) node)
5       (set-dnode-next! (dlist-sentinel xx) node)
6       (inc-dlist-size! dlist))))
7 (define foo (make-dlist))
8 (insert foo 10)
```



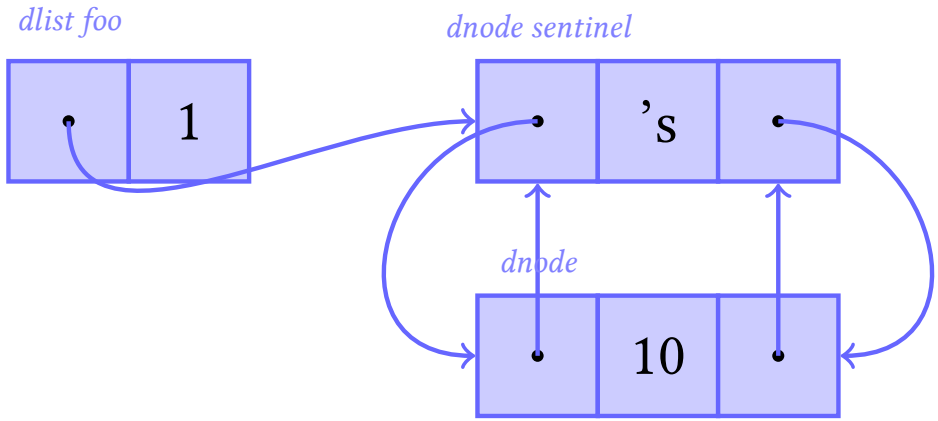
Insert Front

```
1 (define (insert-front xx elt)
2   (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)
3     (begin
4       (set-dnode-prev! (dnode-next node) node)
5       (set-dnode-next! (dlist-sentinel xx) node)
6       (inc-dlist-size! dlist))))
7 (define foo (make-dlist))
8 (insert foo 10)
```



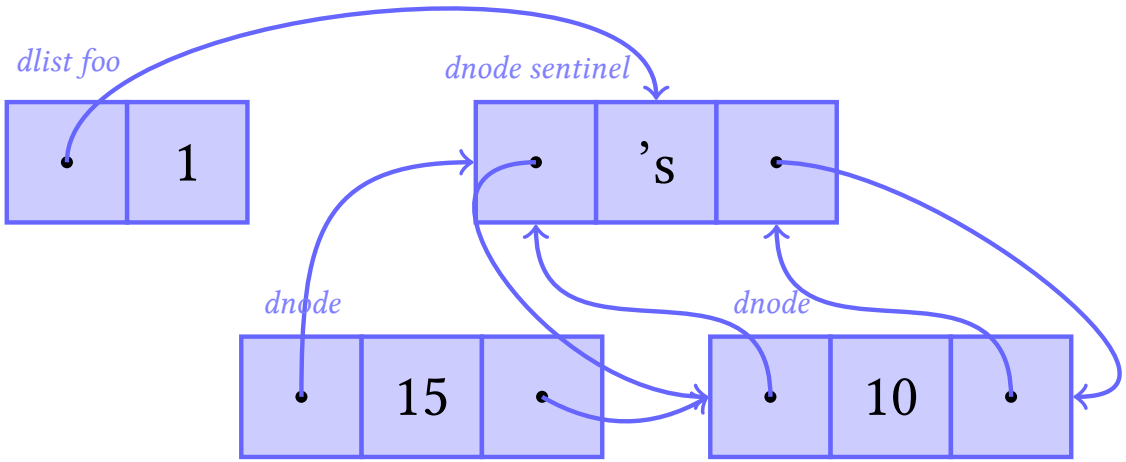
Insert 15

```
(define (insert-front xx elt)
  (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)))))
    (begin
      (set-dnode-prev! (dnode-next node) node)
      (set-dnode-next! (dlist-sentinel xx) node)
      (inc-dlist-size! dlist))))
; ..
(insert foo 15)
```



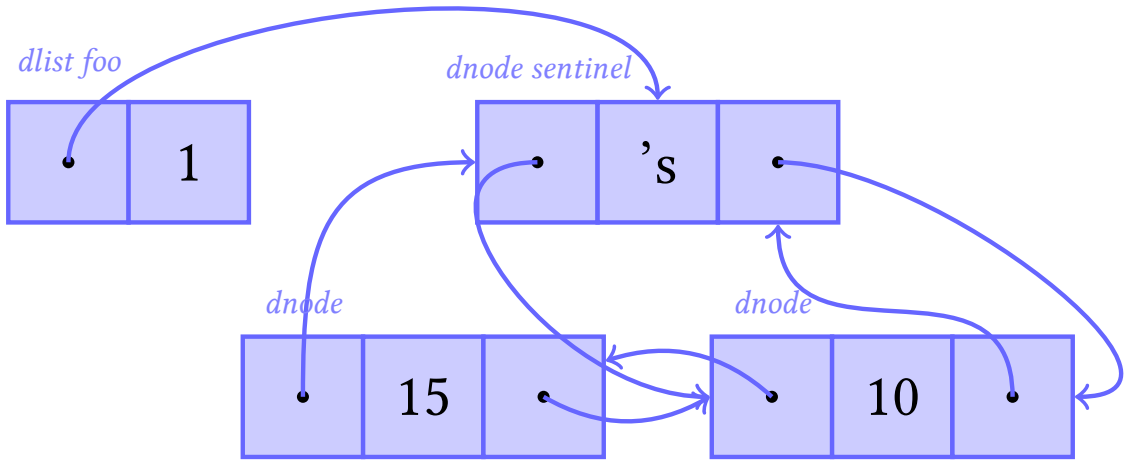
Insert 15

```
(define (insert-front xx elt)
  (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)))))
    (begin
      (set-dnode-prev! (dnode-next node) node)
      (set-dnode-next! (dlist-sentinel xx) node)
      (inc-dlist-size! dlist))))
; ..
(insert foo 15)
```



Insert 15

```
(define (insert-front xx elt)
  (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)))))
    (begin
      (set-dnode-prev! (dnode-next node) node)
      (set-dnode-next! (dlist-sentinel xx) node)
      (inc-dlist-size! dlist))))
; ..
(insert foo 15)
```



Insert 15

```
(define (insert-front xx elt)
  (let ((node (dnode (dlist-sentinel xx) elt (dnode-next (dlist-sentinel xx)))))
    (begin
      (set-dnode-prev! (dnode-next node) node)
      (set-dnode-next! (dlist-sentinel xx) node)
      (inc-dlist-size! dlist))))
; ..
(insert foo 15)
```