



COMPSCI 105 S1 2017

Principles of Computer Science

19 Linked List(3)



Agenda & Readings

▶ Agenda

▶ Variations of Linked Lists

- ▶ Singly Linked Lists with Head and Tail
- ▶ Doubly Linked Lists with Dummy head node

▶ Reference:

▶ Textbook:

- ▶ Problem Solving with Algorithms and Data Structures
 - Chapter 3 – Lists
 - Chapter 3 – The UnorderedList Abstract Data Type

▶ Extra Reading:

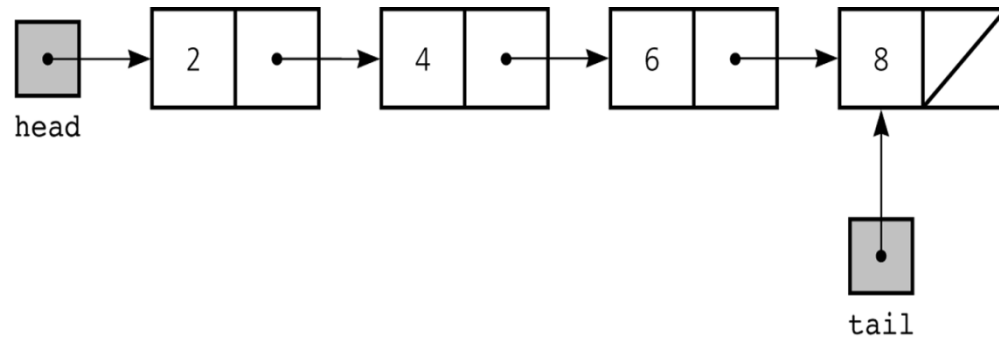
- ▶ [http://en.literateprograms.org/Singly_linked_list_\(Python\)](http://en.literateprograms.org/Singly_linked_list_(Python))



19.1 Singly Linked Lists

Singly Linked List with Head and Tail

- ▶ In some singly linked list implementations, it is handy to have a reference to the last node of the list
 - ▶ Allow **more efficient access** (i.e. insertion) at the **end** of Linked List
 - ▶ Useful for queue-like structure, e.g. a waiting list





19.1 Singly Linked Lists

Singly Linked List with Head and Tail

▶ Cases:

▶ Insertion

▶ General case:

- Insert a new node to the beginning of a list
- Insert a new node to the **end** of a list
- Insert a new node to the middle of a list

▶ An empty list:

- Insert a new node to an **empty** list



19.1 Singly Linked Lists

Singly Linked List with Head and Tail

▶ Cases:

▶ Deletion

▶ General case:

- Remove a node at the beginning of a list
- Remove a node at the **end** of a list
- Remove a node from the middle of a list

▶ Only one node left in the list:

- Remove the **only one node** from a list



19.1 Singly Linked Lists

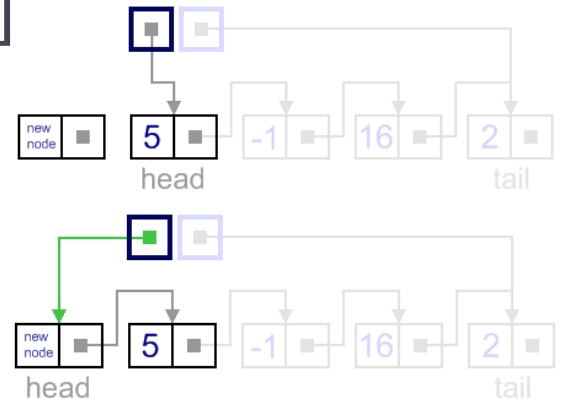
Singly Linked List with Head and Tail

▶ Case 1:

- ▶ Add to head Original linked list: [5, -1, 16, 2]

```
my_list.add_to_head(100)
for num in my_list:
    print(num, end=" ")
```

```
100 5 -1 16 2
```

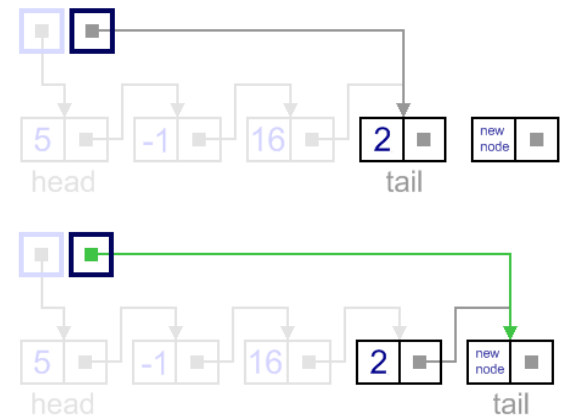


▶ Case 2:

- ▶ Add to tail Original linked list: [5, -1, 16, 2]

```
my_list.add_to_tail(200)
for num in my_list:
    print(num, end=" ")
```

```
5 -1 16 2 200
```





19.1 Singly Linked Lists

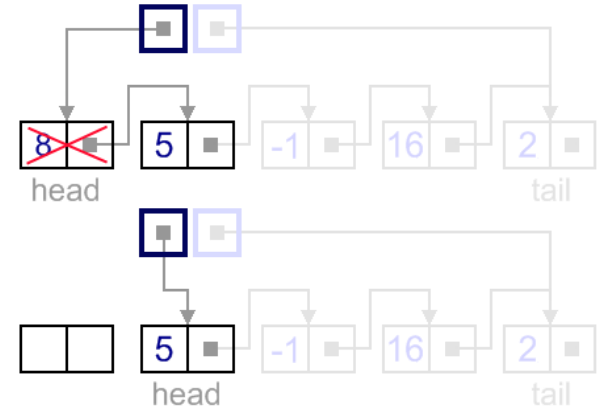
Singly Linked List with Head and Tail

▶ Case 3:

- ▶ Remove from head Original linked list: [8, 5, -1, 16, 2]

```
my_list.remove_from_head()
for num in my_list:
    print(num, end=" ")
```

```
5 -1 16 2
```

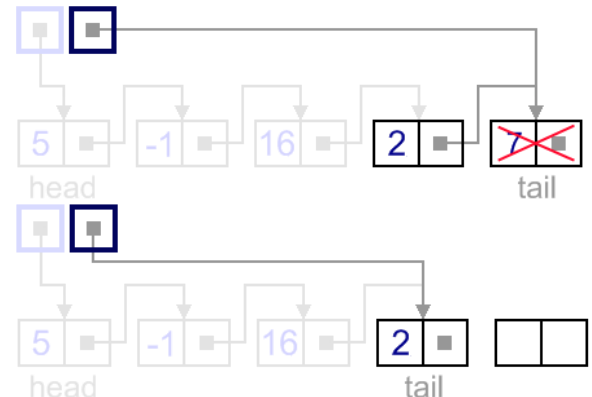


▶ Case 4:

- ▶ Remove from tail Original linked list: [5, -1, 16, 2, 7]

```
my_list.remove_from_tail
for num in my_list:
    print(num, end=" ")
```

```
5 -1 16 2
```



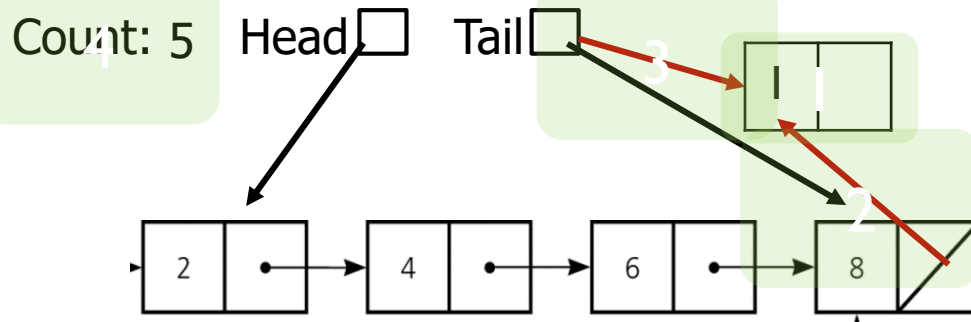


19.1 Singly Linked Lists

Inserting a Node

▶ Add an item to the **end** of the Linked List:

▶ General case: (non-empty list)



```
1 new_node = Node(item)
2 self.tail.set_next(new_node)
3 self.tail = new_node
4 self.count += 1
```

▶ Steps:

- ▶ Create a new node and place the item as its data
- ▶ Change the **next** reference of the old last node of the list to refer to the new node
- ▶ Modify the **tail** to refer to the new node
- ▶ Increase the count

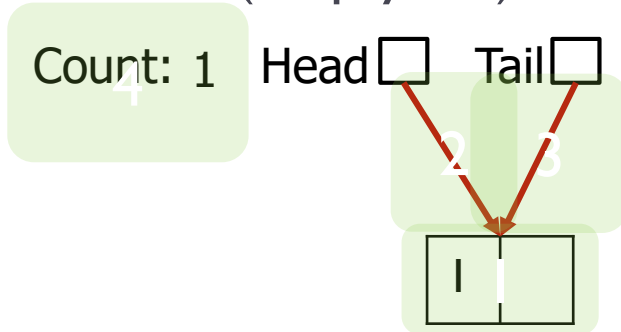


19.1 Singly Linked Lists

Inserting a Node

▶ Add an item to an empty Linked List:

▶ Special case: (empty list)



▶ Steps:

- ▶ Create a new node and place the item as its data
- ▶ Change both the **head** and **tail** to refer to the new node
- ▶ Increase the count

```
1 new_node = Node(item)
2 self.head = new_node
3 if self.tail == None:
4     self.tail = new_node
5 self.count += 1
```

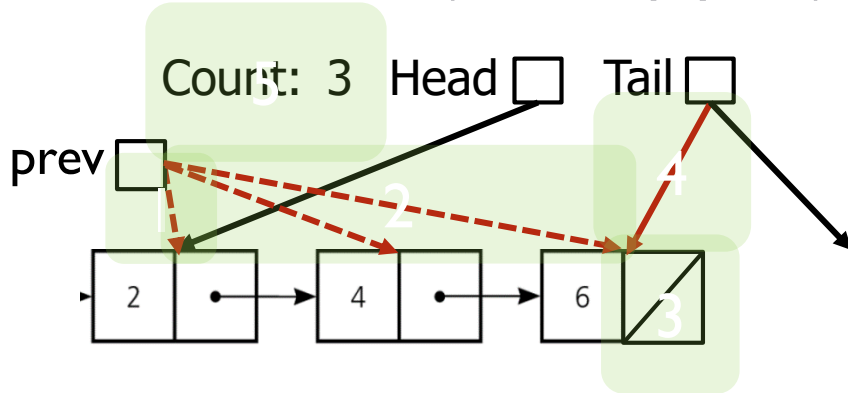


19.1 Singly Linked Lists

Deleting a Node

▶ Remove an item from the **end** of the Linked List:

▶ General case (non-empty list):



```
1 prev = self.head
2 while (prev.get_next() != self.tail):
3     prev = prev.get_next()
4 prev.set_next(None)
5 self.tail = prev
6 self.count -= 1
```

▶ Steps:

- ▶ Locate the previous node
- ▶ Modify the **next** of the pervious node to None
- ▶ Modify the **tail** to refer to the previous node
- ▶ Decrease the count

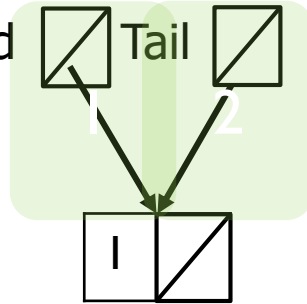


19.1 Singly Linked Lists

Deleting a Node

- ▶ Remove a node from a list with one element

Count: 0 Head



```
1 self.head = self.head.get_next()
2 if self.head == None:
3     self.tail = None
4 self.count -= 1
```

- ▶ Steps:
 - ▶ Modify the head and tail to refer to None
 - ▶ Decrease the count



19.1 Singly Linked Lists

Time Complexity

► Summary

	Python List	Singly Linked List
is_empty	$O(1)$: <i>if len(my_list) == 0</i>	$O(1)$
size	$O(1)$: <i>len(my_list)</i>	$O(1)$ with count variable $O(n)$ without count variable
addTohead	$O(n)$: <i>insert(0, item)</i>	$O(1)$
addToTail	$O(1)$: <i>append</i>	$O(1)$ with tail reference $O(n)$ without tail
add	$O(n)$: <i>insert(index, item)</i>	$O(n)$
removeFromHead	$O(n)$: <i>pop(0)</i>	$O(1)$
removeFromTail	$O(1)$: <i>pop()</i>	$O(n)$ even with tail
remove	$O(n)$: <i>pop(n)</i>	$O(n)$

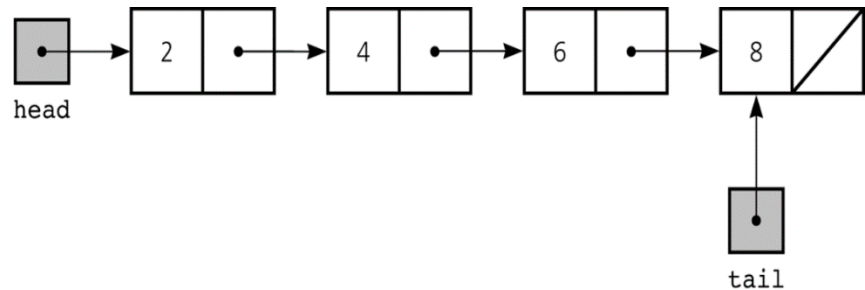


19.1 Singly Linked Lists

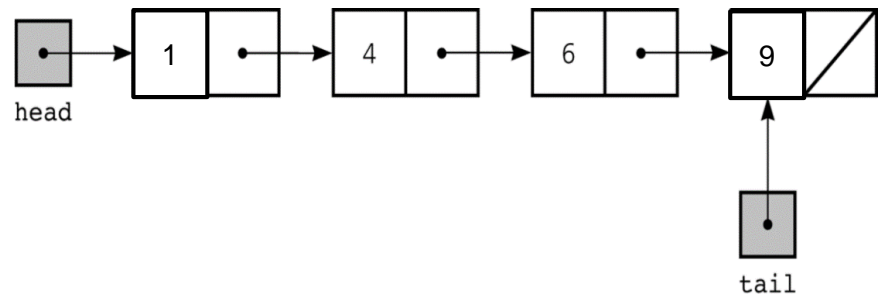
Examples

- ▶ A singly linked list with head and tail references:

```
my_list = LinkedList()
my_list.add_to_head(8)
my_list.add_to_head(6)
my_list.add_to_head(4)
my_list.add_to_head(2)
```



```
my_list.remove_from_tail()
my_list.add_to_tail(9)
my_list.remove_from_head()
my_list.add_to_head(1)
```





Exercise 1

- ▶ What is the output of the following program?

```
my_list = LinkedList()
for x in [9, 2, 22, 40]:
    my_list.add_to_head(x)
my_list.remove_from_tail()
my_list.add_to_tail(32)
my_list.remove_from_head()
my_list.remove_from_head()
my_list.add_to_head(100)
for num in my_list:
    print(num, end=" ")
```



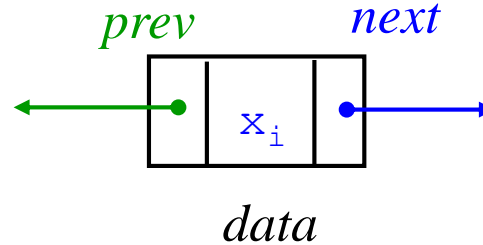
Doubly Linked List

- ▶ A common variation on linked lists is to have two pointers to other nodes within each node:
 - ▶ One to the **next** node on the list
 - ▶ One to the **previous** node
- ▶ Doubly-linked lists make some operations, such as deleting a tail node, more efficient
- ▶ Double-linked lists can have iterators for efficient **forward** and **backward** traversals



Doubly Linked List

- ▶ Each **NodeDLL** references both its predecessor and its successor



```
class NodeDLL:  
    def __init__(self, init_data, next_node=None, prev_node=None):  
        ...
```

- ▶ Each object in a doubly linked list will contain three member variables:
 - ▶ **data**: value stored in this node
 - ▶ **next**: refers to the next node in the list
 - ▶ **prev**: refers to the previous node in the list



19.2 Doubly Linked Lists

NodeDLL Class

► Code:

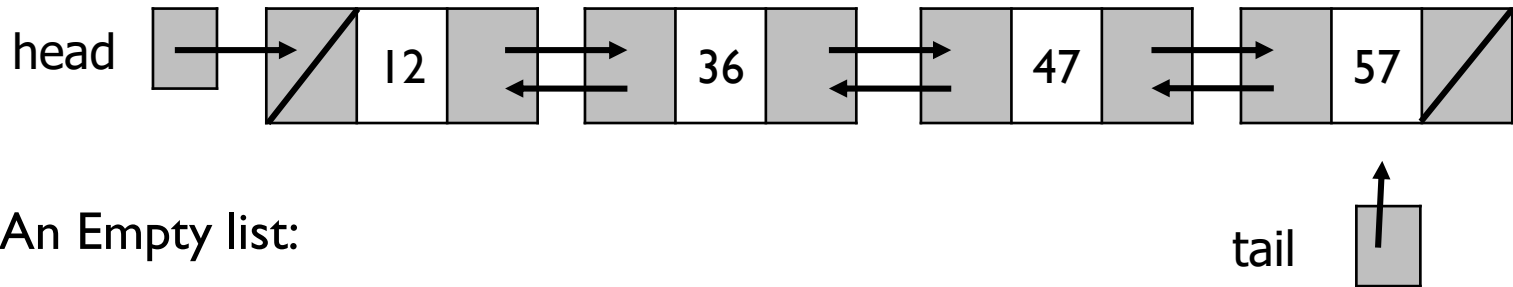
```
class NodeDLL:
    def __init__(self, init_data, next_node=None, prev_node=None):
        ...
    def get_data(self):
        return self.data
    def get_prev(self):
        return self.prev
    def get_next(self):
        return self.next
    def set_data(self, new_data):
        self.data = new_data
    def set_next(self, new_next):
        self.next = new_next
    def set_prev(self, new_prev):
        self.prev = new_prev
    ...
```



Doubly Linked List

▶ Doubly Linked List

- ▶ A head reference is used to reference the first node in the list
- ▶ A tail reference points to the last node.
- ▶ Examples:
 - ▶ A doubly linked list with 4 nodes:



- ▶ An Empty list:

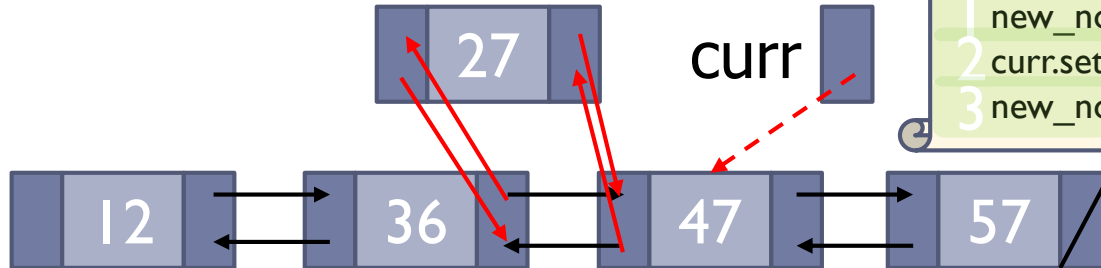




Inserting a Node

▶ Add an item to the middle of the Linked List:

▶ General case: (non-empty list)



```
def add_before(self, item, curr):  
    1 new_node = NodeDLL(item, curr, curr.get_prev())  
    2 curr.set_prev(new_node)  
    3 new_node.get_prev().set_next(new_node)
```

▶ Steps:

- ▶ Create a new node and place the item as its data
 - Set the **next** of the new node to refer to the **curr**
 - Set the **prev** of the new node to refer to the previous node of **curr**
- ▶ Modify the **prev** of the **curr** node to refer to the new node
- ▶ Modify the next of the node that is to precede the new node to refer to the new node



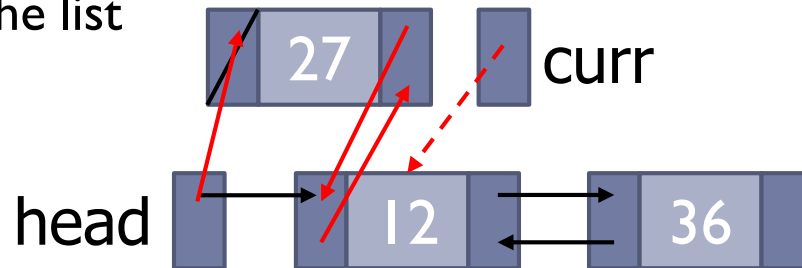
19.2 Doubly Linked Lists

Inserting a Node

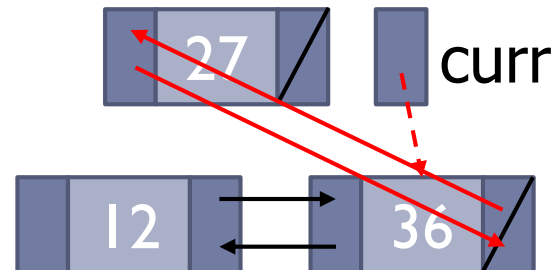
- ▶ Add an item to the Linked List:
 - ▶ However, we still need to handle insertion in different cases:
 - ▶ Case 1: an empty list



- ▶ Case 2: at the beginning of the list



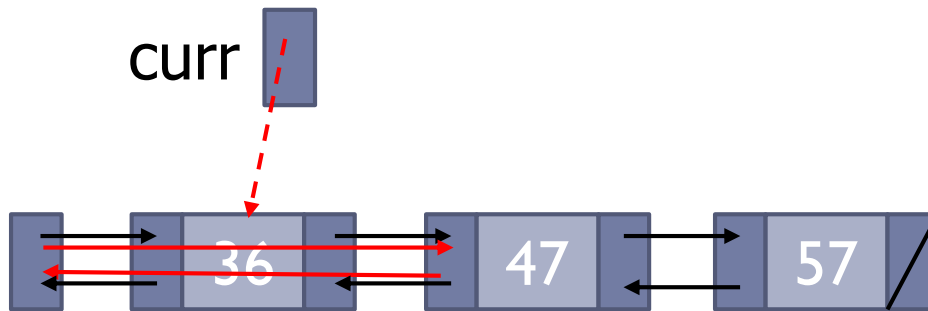
- ▶ Case 3: at the end of the list





Deleting a Node

- ▶ Remove an item from the middle of the Linked List:
 - ▶ General case: (non-empty list)



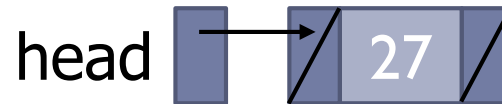
```
def remove(self, curr):  
1 curr.get_prev().set_next(curr.get_next())  
2 curr.get_next().set_prev(curr.get_prev())
```

- ▶ Steps:
 - ▶ Modify the **next** of the node that precede **curr** so that it refers to the node that follows **curr**
 - ▶ Modify the **prev** of the node that follows **curr** so that it refers to the node that precedes **curr**

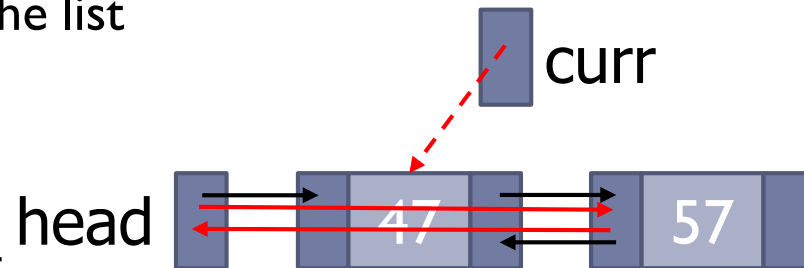


Deleting a Node

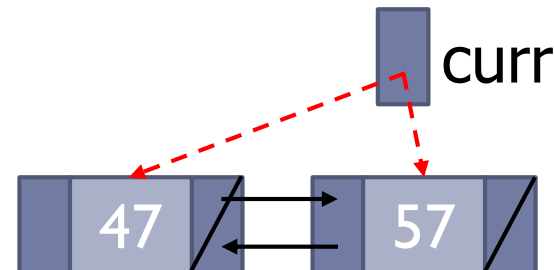
- ▶ Remove an item from the the Linked List
 - ▶ However, we still need to handle insertion in different cases:
 - ▶ Case 1: One element



- ▶ Case 2: at the beginning of the list



- ▶ Case 3: at the end of the list



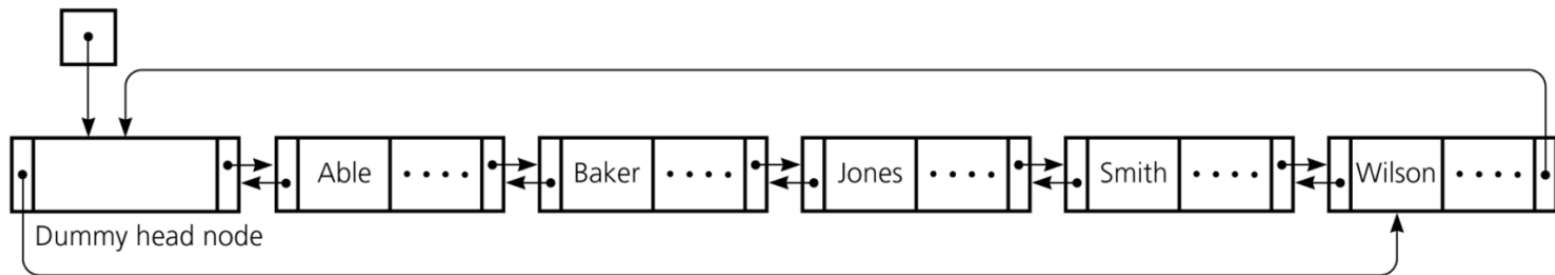


19.2 Doubly Linked Lists

Circular + Dummy Head Node

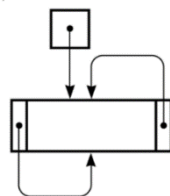
- ▶ How can we simplify the implementation?
 - ▶ Use Circular doubly linked list with a dummy head node
 - ▶ The prev of the dummy head node refers to the last node
 - ▶ The next of the last node refers to the dummy head node

(a) listHead



- ▶ Eliminates special cases for insertions and deletions
- ▶ Dummy head node is always present, even when the linked list is empty

(b) listHead

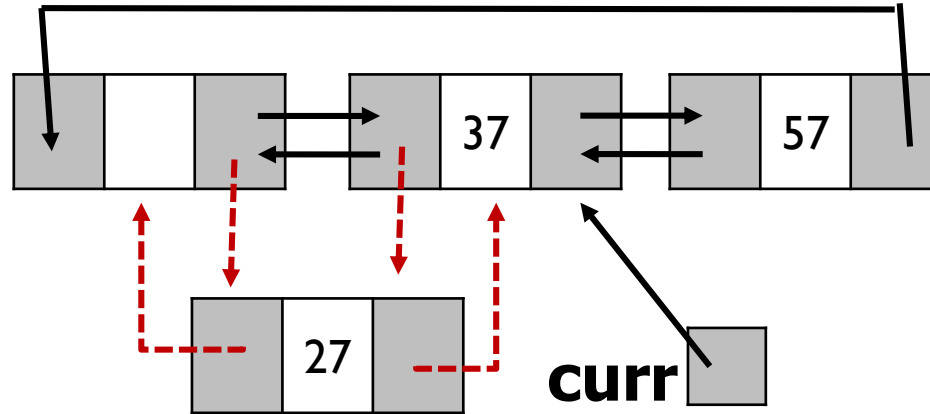




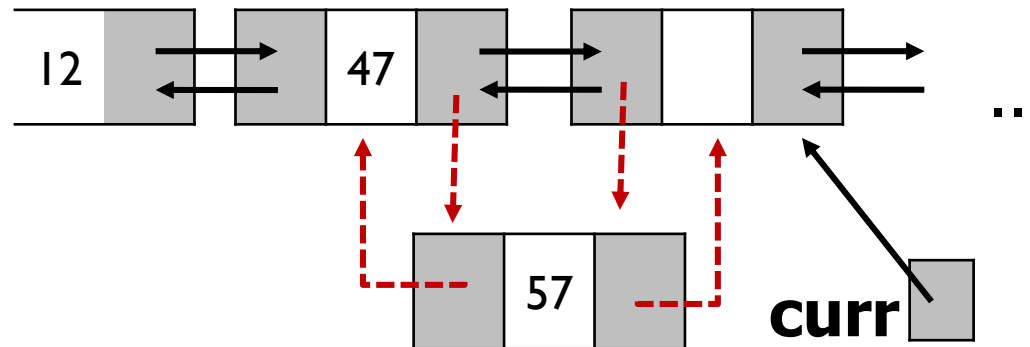
Circular + Dummy Head Node

▶ Inserting a Node

▶ At the beginning:



▶ At the end:

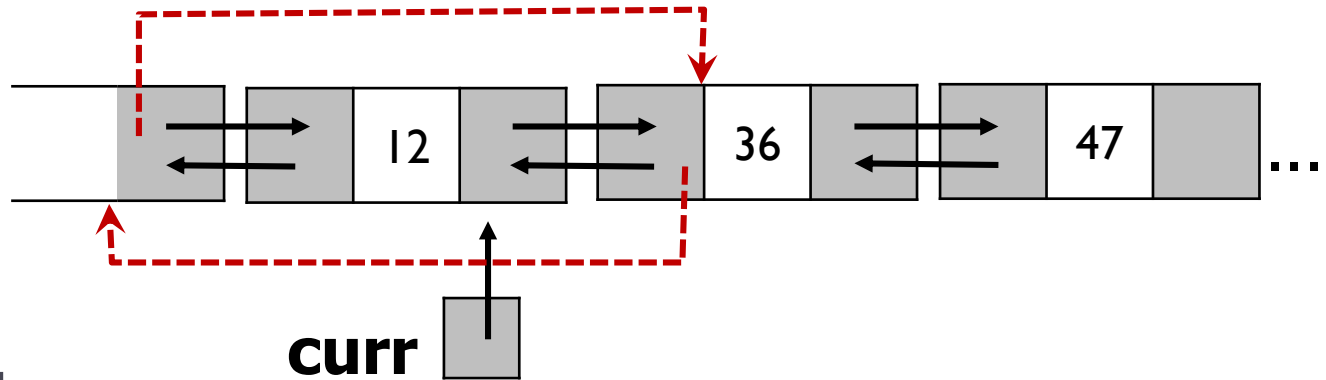




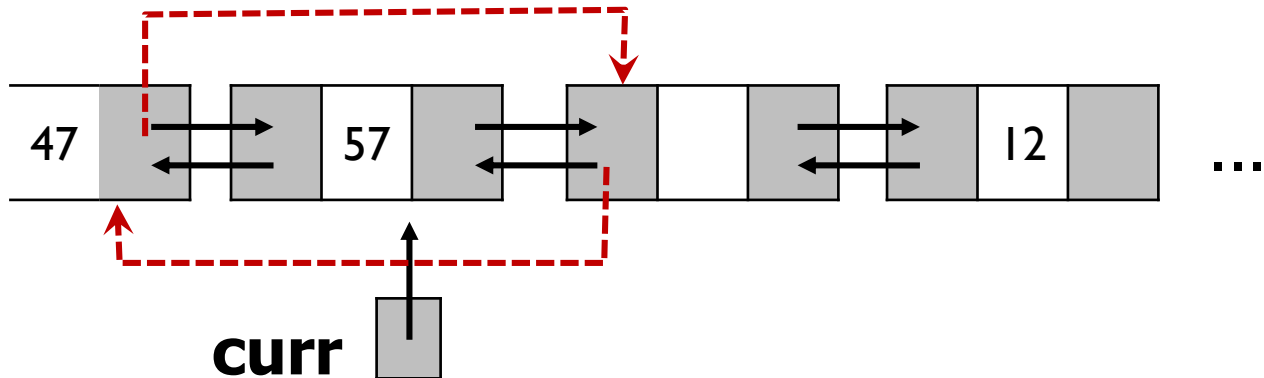
Circular + Dummy Head Node

▶ Deleting a Node

▶ At the beginning:



▶ At the end:





Time Complexity

► Summary

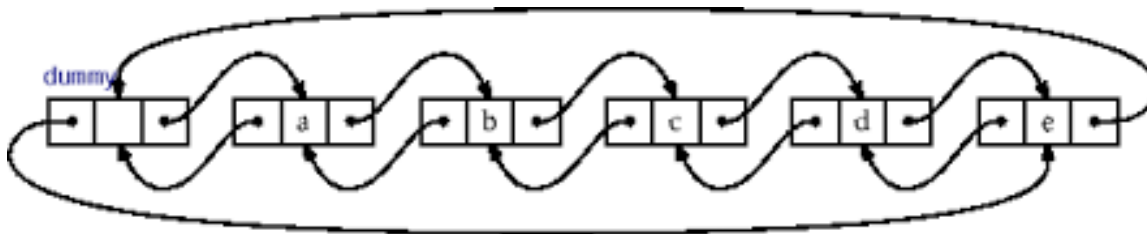
	Python List	Singly Linked List	Circular Doubly with a dummy head
is_empty	$O(1)$	$O(1)$	$O(1)$
size	$O(1)$	$O(1)$ with count	$O(1)$ with count
addTohead	$O(n)$	$O(1)$	$O(1)$
addToTail	$O(1)$	$O(1)$ with tail $O(n)$ without	$O(1)$
add	$O(n)$	$O(n)$	$O(n)$



Exercise 2

- ▶ Consider the following code fragment

```
from LinkedListDLL import LinkedListDLL
def Test_DoublyLinkedList():
    my_list.add_to_tail('b')
    my_list.add_to_tail('c')
    my_list.add_to_head('a')
    my_list.add_to_tail('d')
    my_list.add_to_tail('e')
```





Exercise 2

- ▶ Consider the following code fragment

```
from LinkedListDLL import LinkedListDLL
def Test_DoublyLinkedList():
    my_list.add_to_tail(2)
    my_list.add_to_tail(4)
    my_list.add_to_head(6)
    my_list.add_to_tail(8)
```

- ▶ Draw the resulting doubly linked list in the space provided



Summary

- ▶ Understand and learn how to implement the singly linked-list
- ▶ Understand and learn how to implement the doubly linked-list