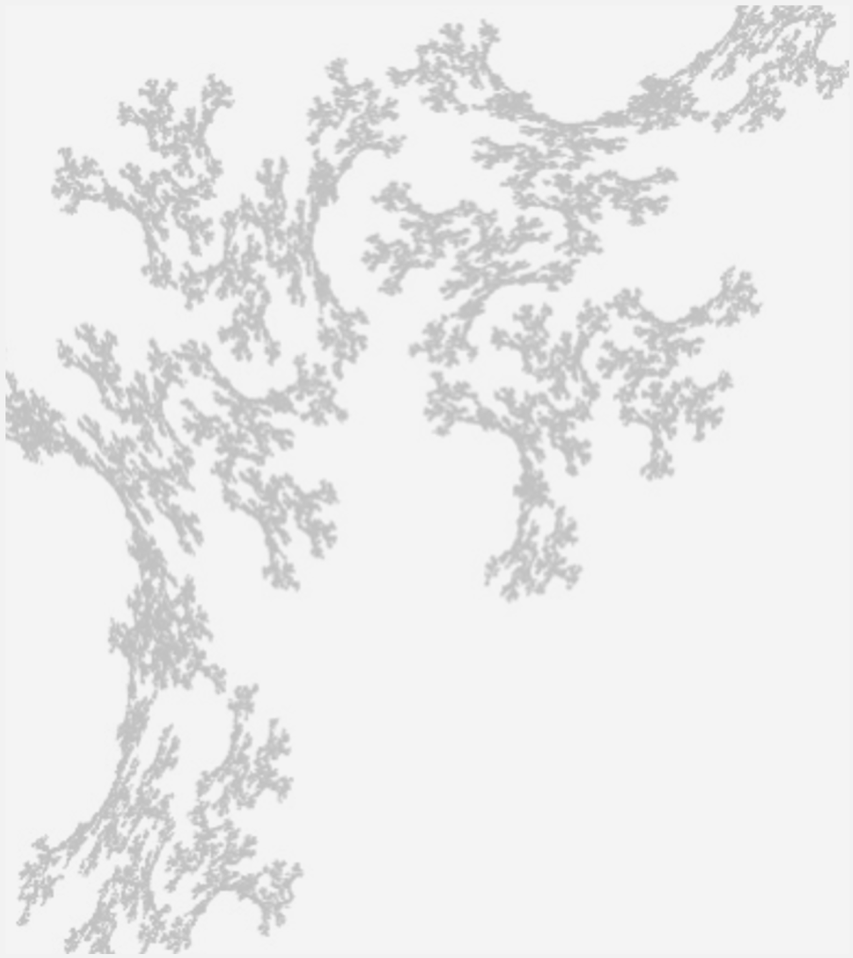
03. Special Linked Structures

- *Introduction*
- *Linked Lists*
- *Doubly Linked Lists*
- *Circular Linked Lists*



03. Special linked Structures

- *Introduction*
- *Singly Linked Lists (revisit)*
- *Doubly Linked Lists*
- *Circular Linked Lists*

Introduction

Array. A collection of contiguous data elements whose order is given by the physical memory location.

Linked List. A collection of data elements whose order is NOT given by the physical memory location.

Arrays and Linked Lists are the most fundamental memory organizations that are used to create more data structure abstractions such as Stacks and Queues.

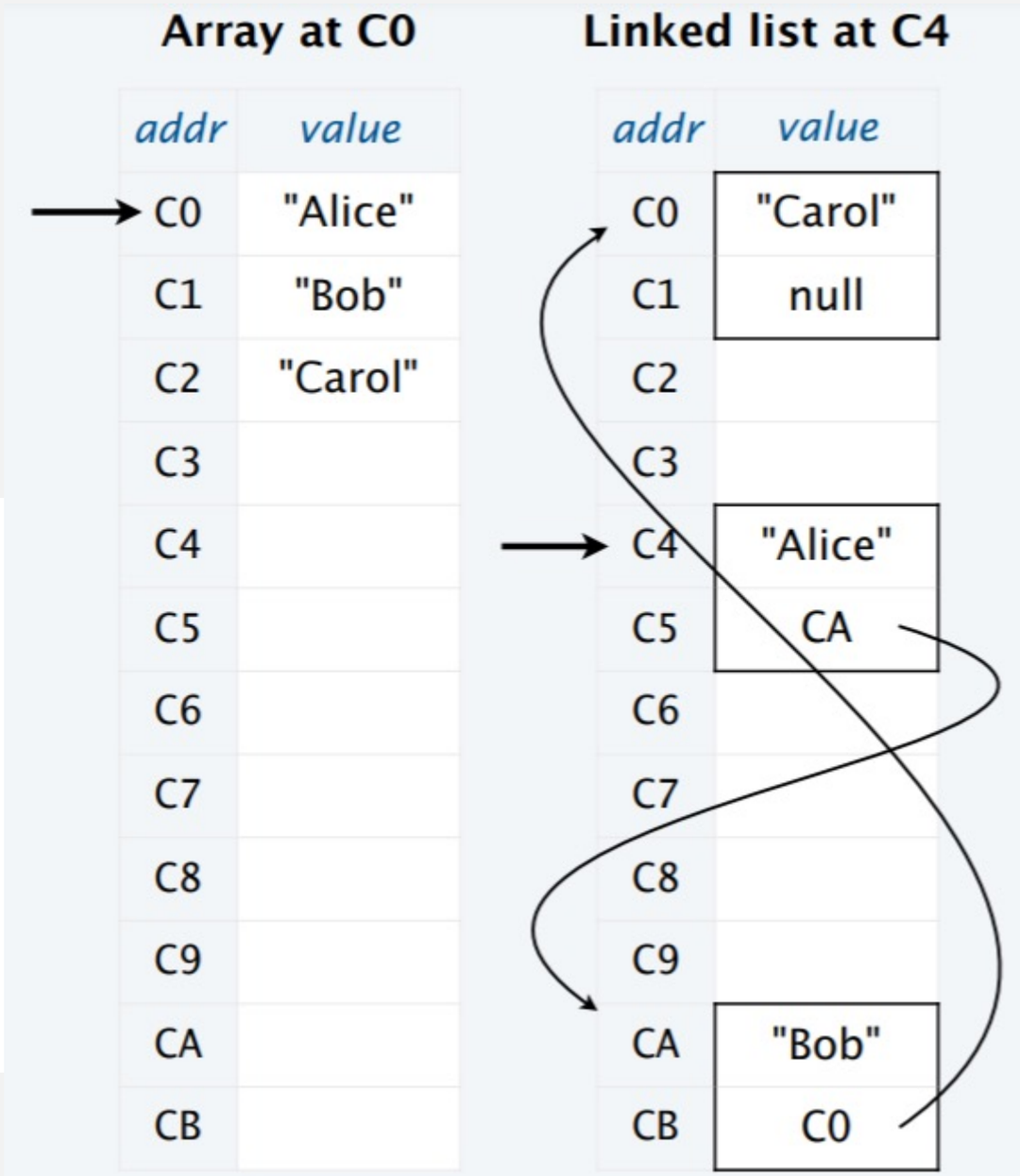
Sequential and Linked Data Structures

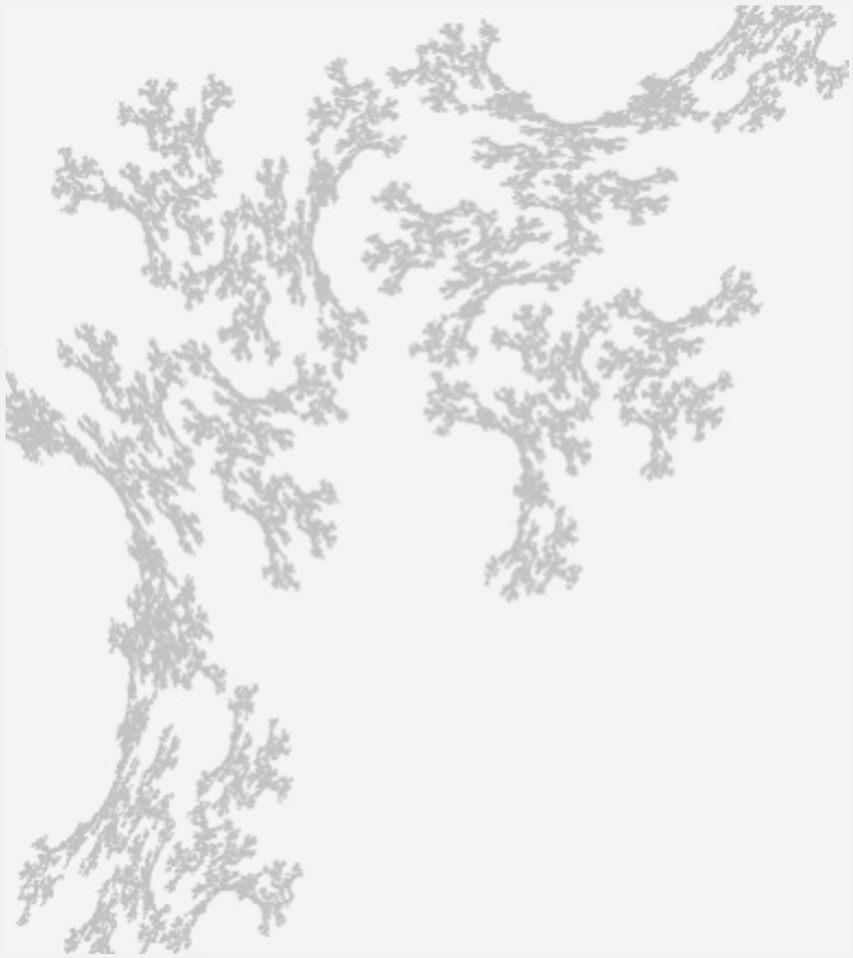
Sequential data structure

- Put objects next to one another.
- Machine: consecutive memory cells.
- Java: array of objects.
- Fixed size, arbitrary access. ⟵ *i*th element

Linked data structure

- Associate with each object a link to another one.
- Machine: link is memory address of next object.
- Java: link is reference to next object.
- Variable size, sequential access. ⟵ *next* element
- Overlooked by novice programmers.
- Flexible, widely used method for organizing data.





Special linked Structures

- *Introduction*
- *Singly Linked Lists (revisited)*
- *Doubly Linked Lists*
- *Circular Linked Lists*

Simplest singly-linked data structure – Linked List

Linked list

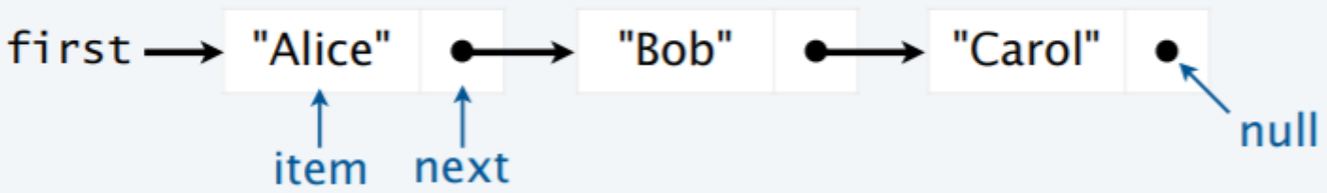
- A recursive data structure.
- Def. A *linked list* is null or a reference to a *node*.
- Def. A *node* is a data type that contains a reference to a node.
- Unwind recursion: A linked list is a sequence of nodes.

```
private class Node
{
  private String item;
  private Node next;
}
```

Representation

- Use a private nested class Node to implement the node abstraction.
- For simplicity, start with nodes having two values: a String and a Node.

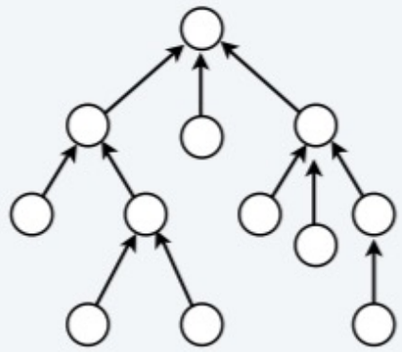
A linked list



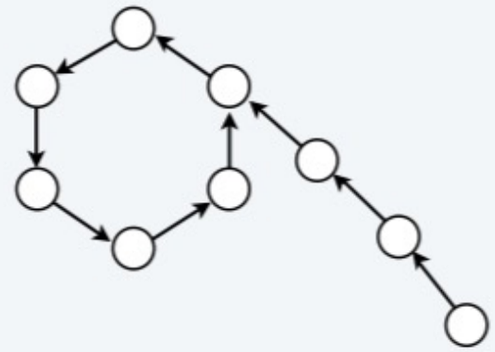
Linked Data Structures

Linked list (this lecture)

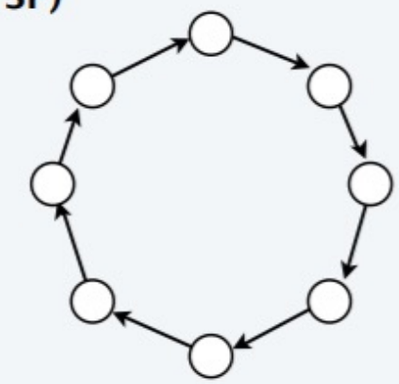
Tree



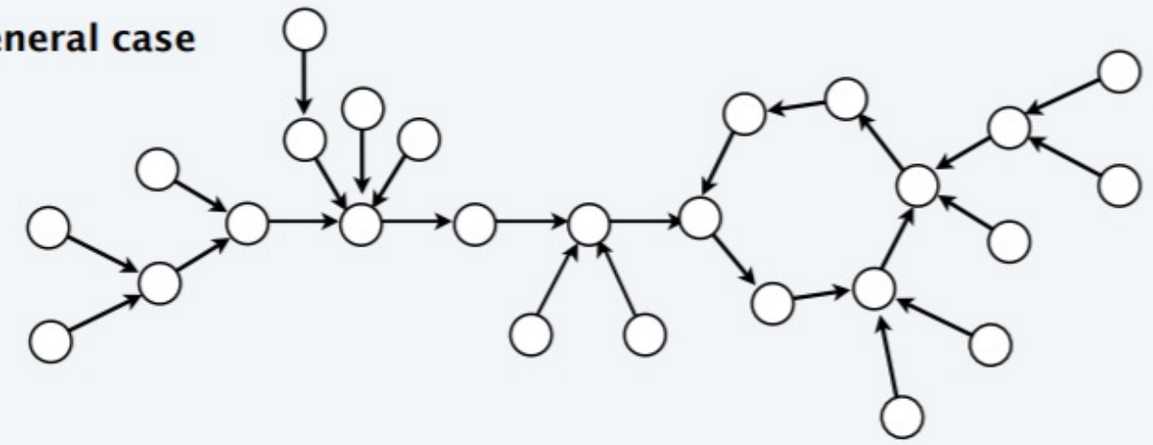
Rho



Circular list (TSP)



General case



Multiply linked structures: many more possibilities!

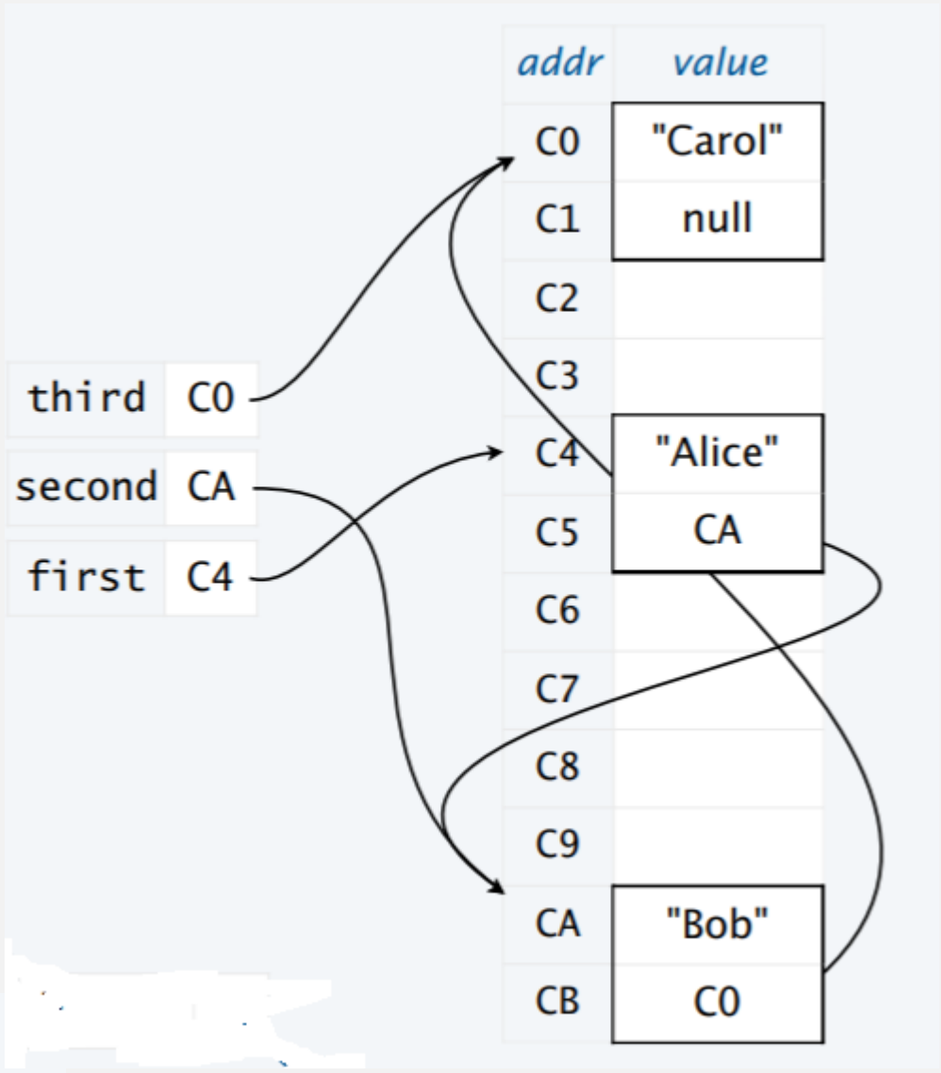
From the point of view of a particular object, all of these structures look the same.

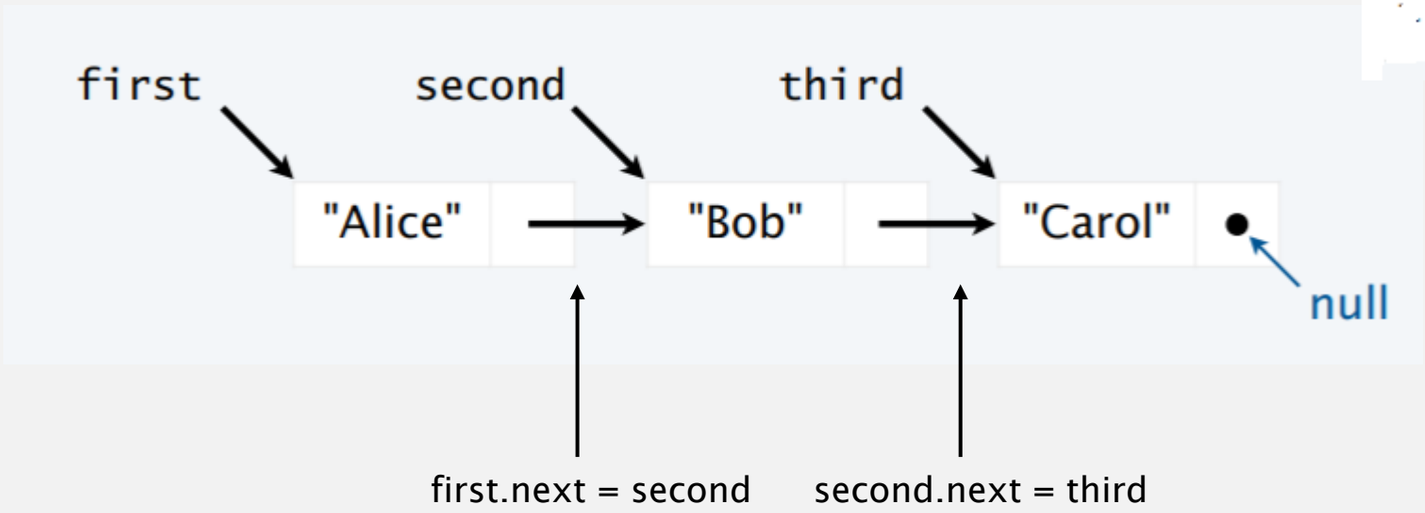
Building a Singly Linked List

```
Node first  = new Node();
first.item  = "Alice";
first.next  = second;
```

```
Node second = new Node();
second.item = "Bob";
second.next = third;
```

```
Node third  = new Node();
third.item  = "Carol";
third.next  = null;
```

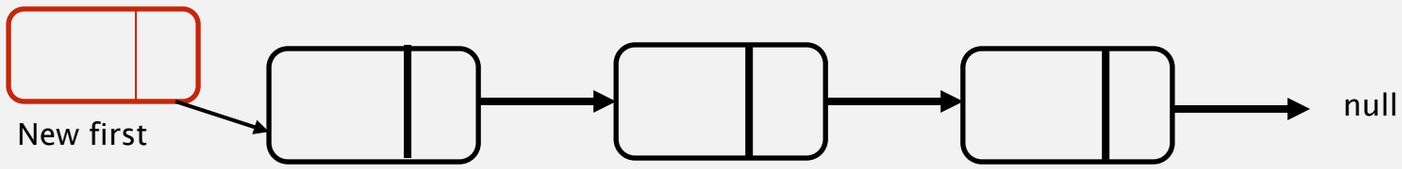




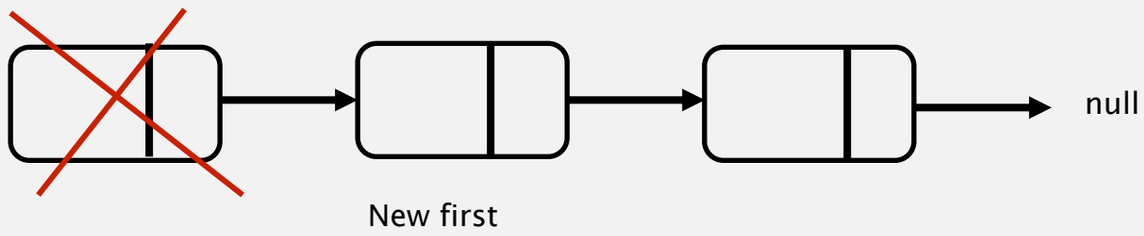
List Processing Code

Standard operations for processing data structured as a singly-linked list

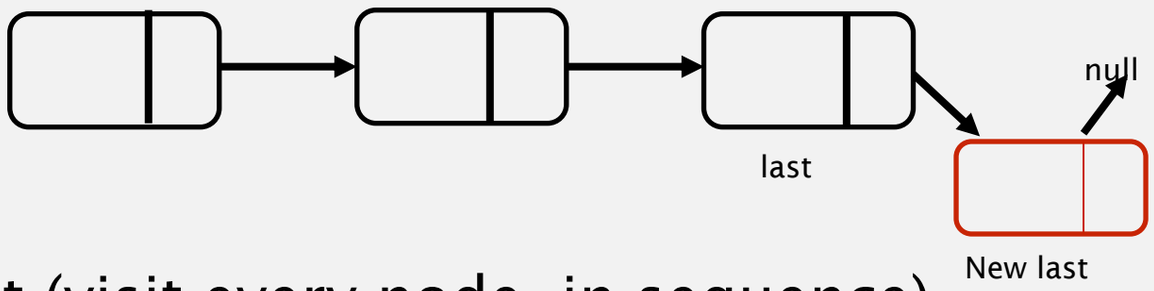
- Add a node at the beginning of the list



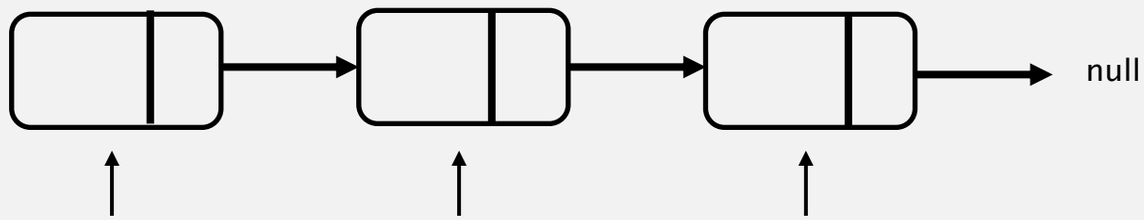
- Remove and return the node at the beginning



- Add a node at the end (requires a reference to the last node)



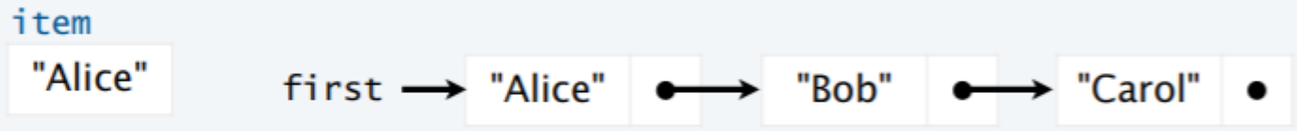
- Traverse the list (visit every node, in sequence).



List Processing code – Remove and Return First Item

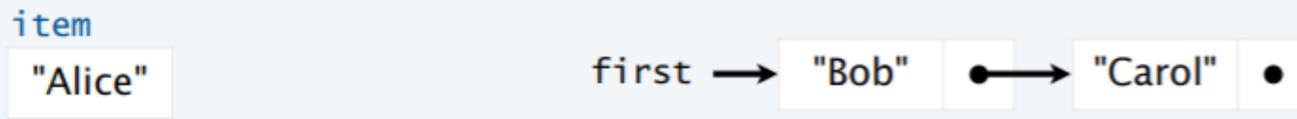
Goal. Remove and return the first item in a linked list `first`.


```
item = first.item;
```




```
first = first.next;
```


```
return item;
```

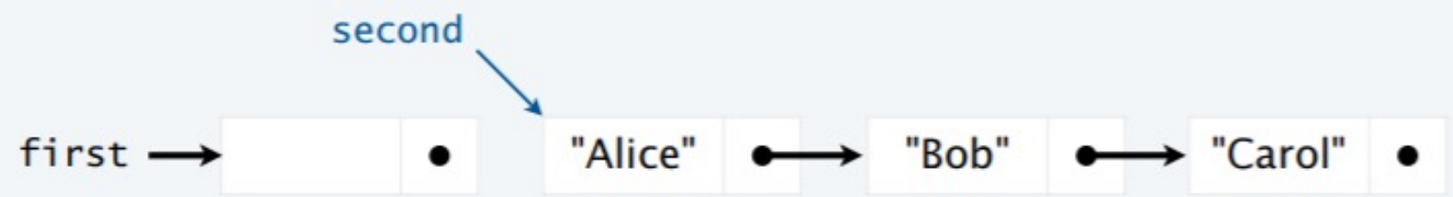


List Processing Code – Add a new node at the beginning

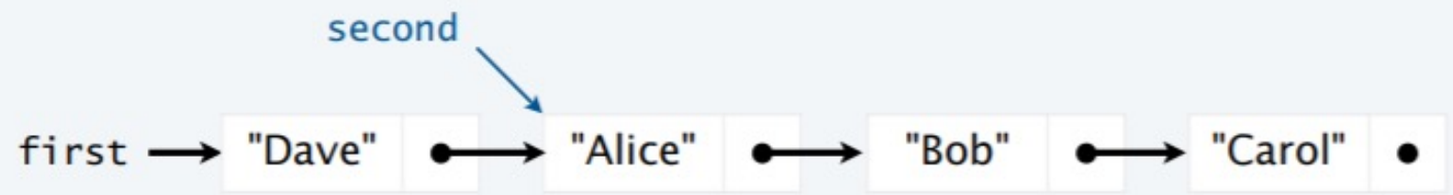
Goal. Add `item` to a linked list `first`.


```
Node second = first;
```


```
first = new Node();
```



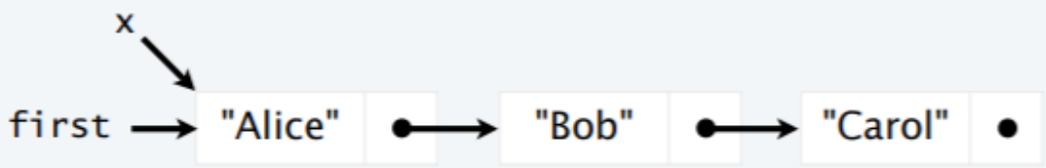

```
first.item = item;
first.next = second;
```



List processing code – Traverse a List

Goal. Visit every node on a linked list `first`.

```
Node x = first;
while (x != null)
{
   StdOut.println(x.item);
   x = x.next;
}
```



StdOut

```
Alice
Bob
Carol
```

Pop quiz1 on Linked Lists

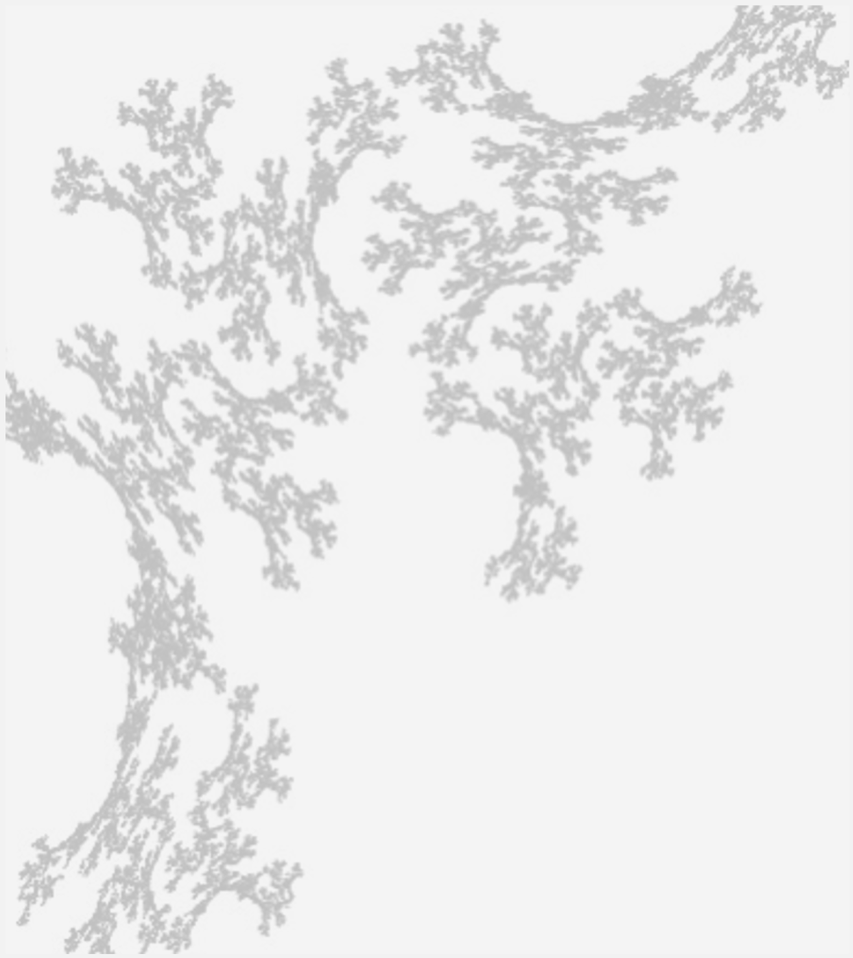
Q. What is the effect of the following code (not-so-easy question)?

```
...
Node list = null;
while (!StdIn.isEmpty())
{
   Node old = list;
   list = new Node();
   list.item = StdIn.readString();
   list.next = old;
}
for (Node t = list; t != null; t = t.next)
   StdOut.println(t.item);
...
```

Pop quiz 2 on Linked Lists

Q. What is the effect of the following code (not-so-easy question)?

```
...
Node list = new Node();
list.item = StdIn.readString();
Node last = list;
while (!StdIn.isEmpty())
{
   last.next = new Node();
   last = last.next;
   last.item = StdIn.readString();
}
...
```

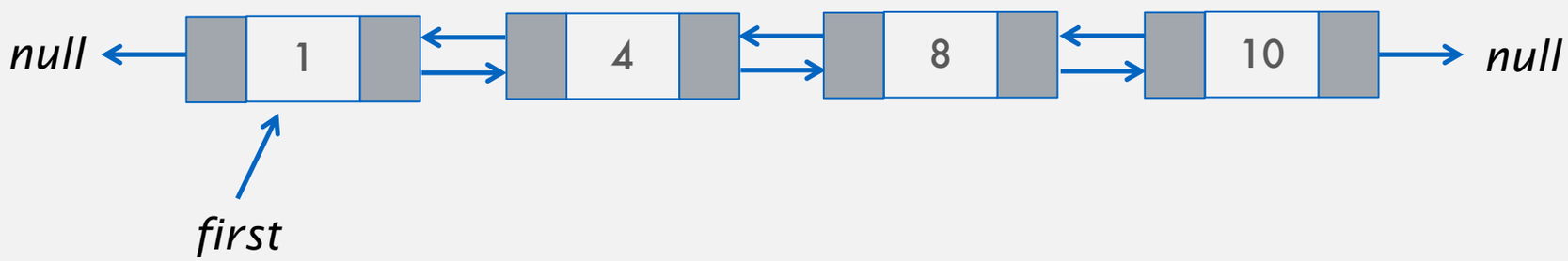


Special linked Structures

- *introduction*
- *Singly Linked Lists*
- *Doubly Linked Lists*
- *Circular Linked Lists*

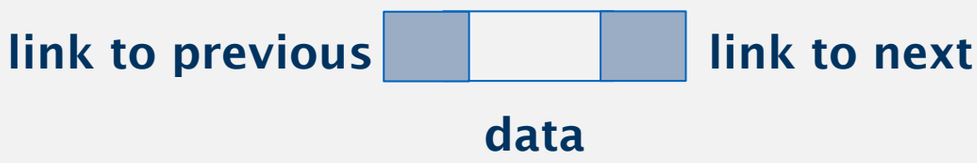
Doubly Linked Lists

A linked list where every node refers to its previous and next nodes



Each node has three parts:

- a data part
- a link to the previous node
- a link to the next node



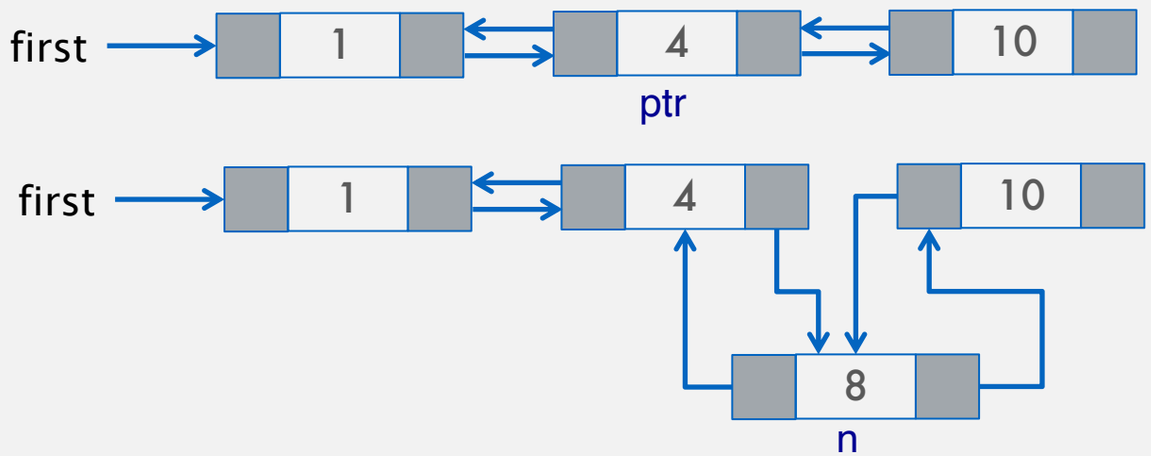

```
class DoublyLinkedList <T> {
    Node first;
    int n;
}
```

```
class Node <T> {
    T data;
    Node prev;
    Node next;
}
```

Implementation of a Doubly Linked List

Insert after a target

Create the new node 8 and insert it after 4



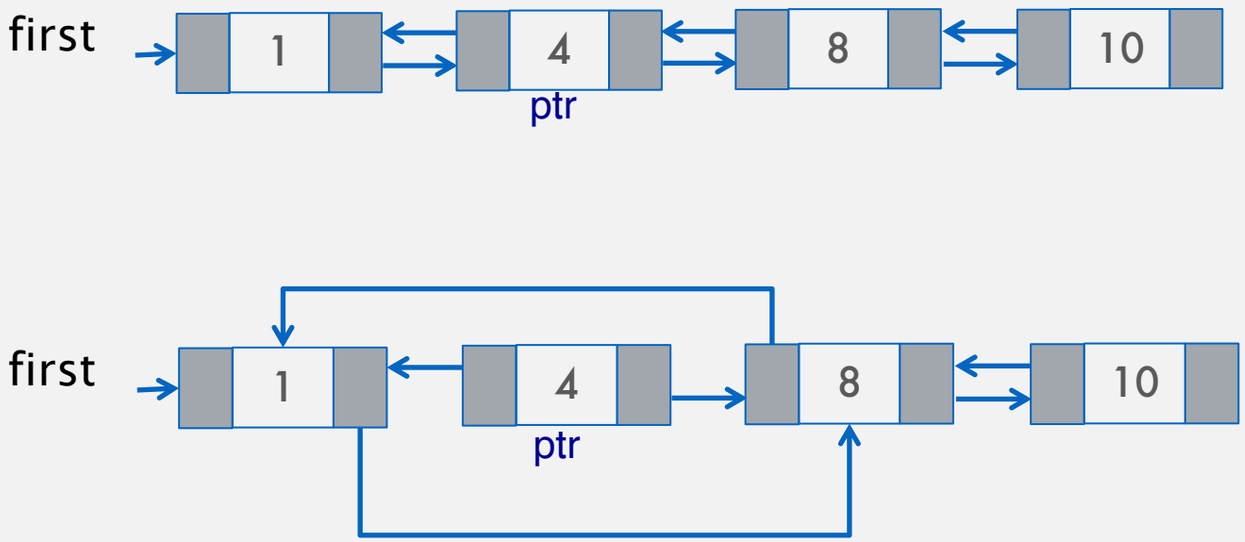

```
Node n        = new Node();
n.data        = 8;
n.next        = ptr.next;
ptr.next      = n;
n.next.prev = n;
n.prev        = ptr;
```

Running time? O(1)

Implementation of a Doubly Linked List

Remove a target

Delete node containing 4




```
ptr.prev.next = ptr.next;
ptr.next.prev = ptr.prev;
```

Running time?

- Best case if target is the first node: O(1)
- Worst case if target is the last node: O(n)

Applications of Doubly Linked Lists

LO 3.4, 3.6

- A doubly linked might be suitable for many applications where it is important to be able to navigate the linked structure in any direction (forward or backward)

- An application is to maintain the history of web pages visited where both backward and forward navigation of the history is possible.

- In some applications it can be used to maintain most and least recently used memory cache.

- One can design a music player with a next and previous buttons to maintain the history of the playlist, both forward and backward.

Advantages and Disadvantages of Doubly Linked Lists

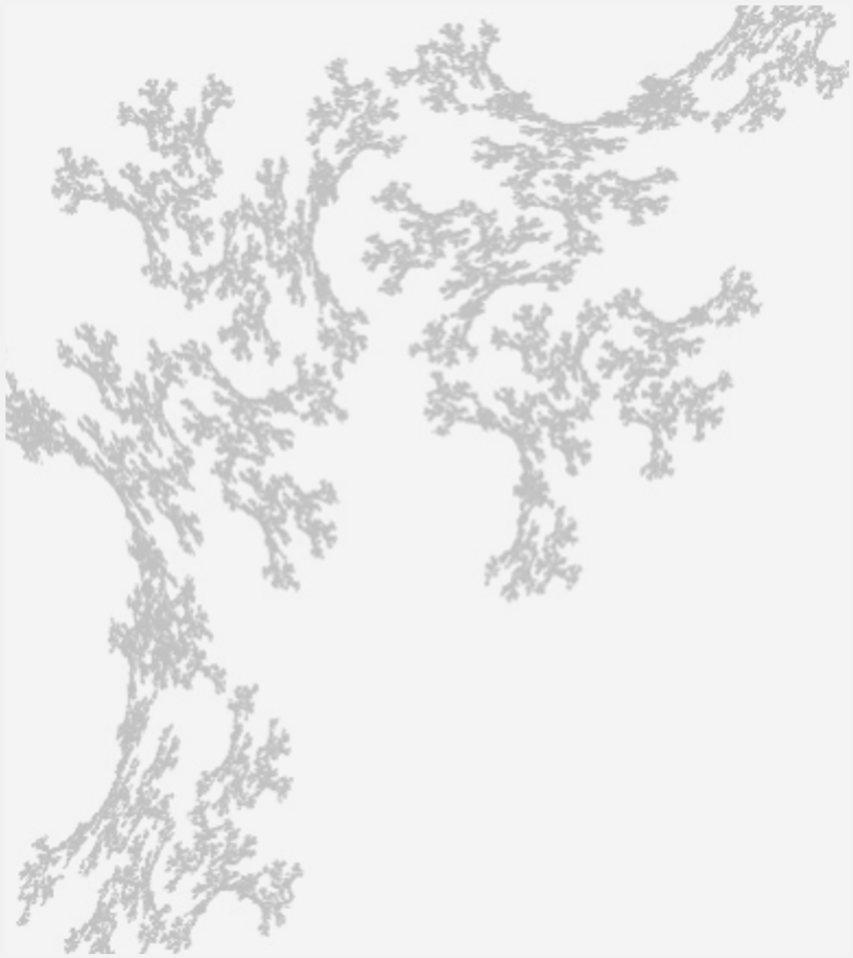
LO 3.4

Disadvantages

- Each node requires extra 8 bytes of memory for previous reference
- It is harder to implement methods as operations such as inserting a node to the middle requires 4 statements

Advantages

- More versatile than singly linked list as one can traverse in both directions.
- When implementing a dynamic application, reference to mostRecentlyAccessednode can be saved in order to move forward or backward.

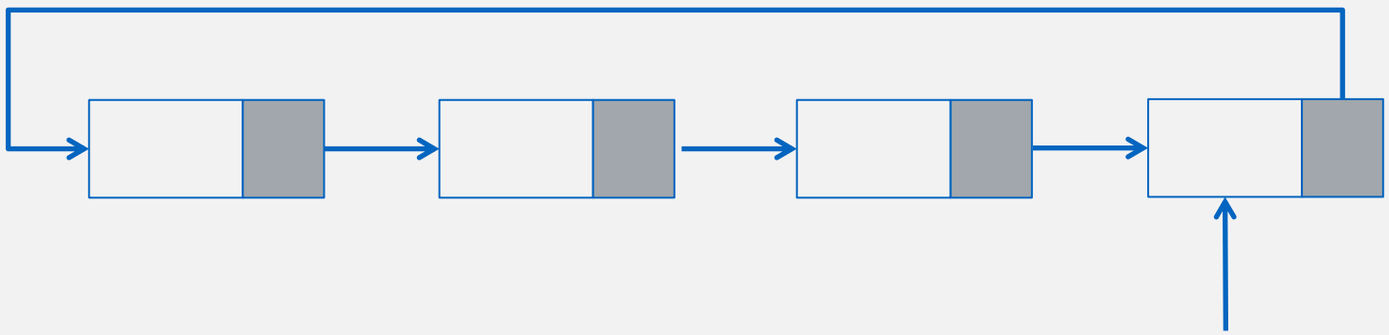


Special linked Structures

- *introduction*
- *Singly Linked Lists*
- *Doubly Linked Lists*
- *Circular Linked Lists*

Circular Linked Lists

A circular link list is a special form of a singly linked list where all nodes are in a continuous cycle. There is no use of null in a circular link list to indicate the end of the list.



By keeping a pointer to the last entry we have access to the *first* and *last* entry in constant time.
last: last
first: last.next

```
class CircularLinkedList <T> {
    Node last;
    int n;
}
```

Circular Linked lists have the property that the last node refers back to the first node: last.next is the reference to the first node

Advantages of Circular Linked Lists

LO 3.4

A circular linked list can be traversed beginning at any node in the list.

A circular linked list can be used to implement a data structure such as a queue where two references front, and back are needed. However, we note that back.next = front

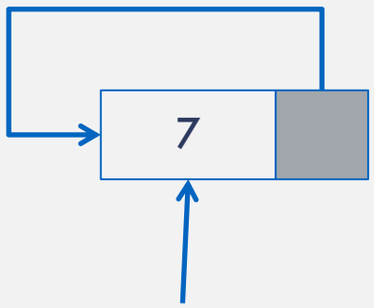
A circular linked list can also be singly or doubly.

Implementation of a circular Linked List

Add to front: add an element to the front of the list

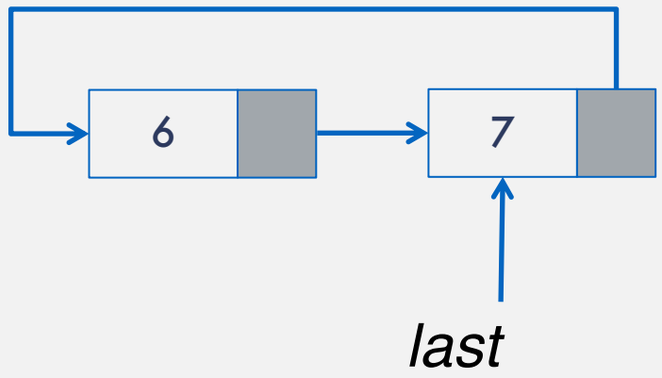
Two cases:

1. List is empty




```
Node n = new Node();
n.data = 7;
n.next = n;
last = n;
```

2. List is not empty




```
Node n = new Node();
n.data = 6;
n.next = last.next;
last.next = n;
```

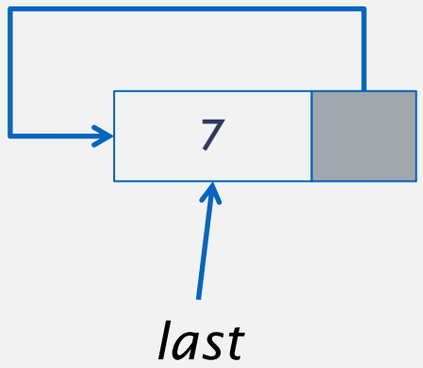
Running time: O(1)

Implementation of a circular Linked List

Remove front: deletes the first element of the list

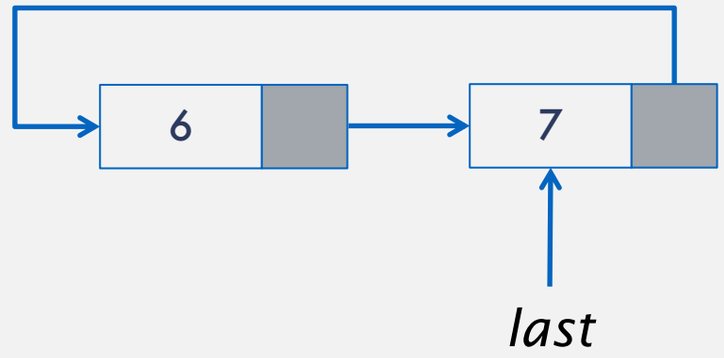
Three cases:

1. List is empty
2. One element

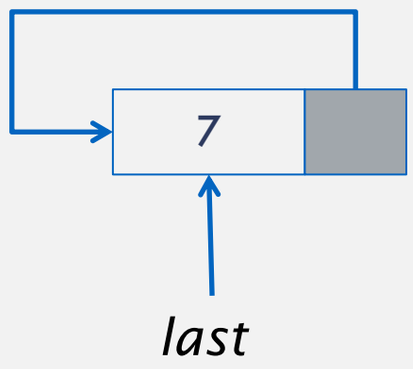



```
last = null;
```

3. More than one element




```
last.next = last.next.next;
```

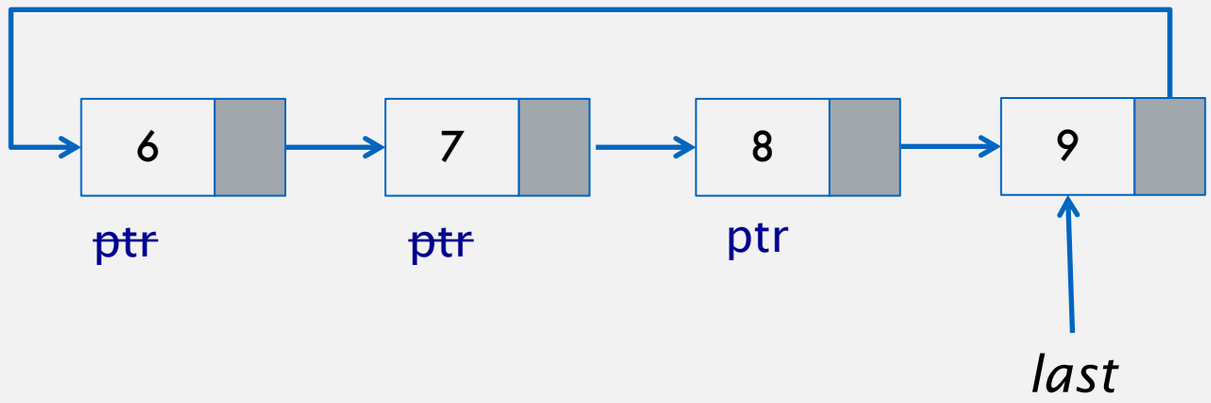


Running time: O(1)

Implementation of a circular Linked List

To search in a CLL: say target is 8

1. Start a pointer at the front
2. Advance pointer until target is found or the beginning of the list is reached again.




```
Node ptr = last.next;
do{
    if (ptr.data == target) {
     break;
    }
    ptr = ptr.next;
} while (ptr != last.next);
```

What is the running time?

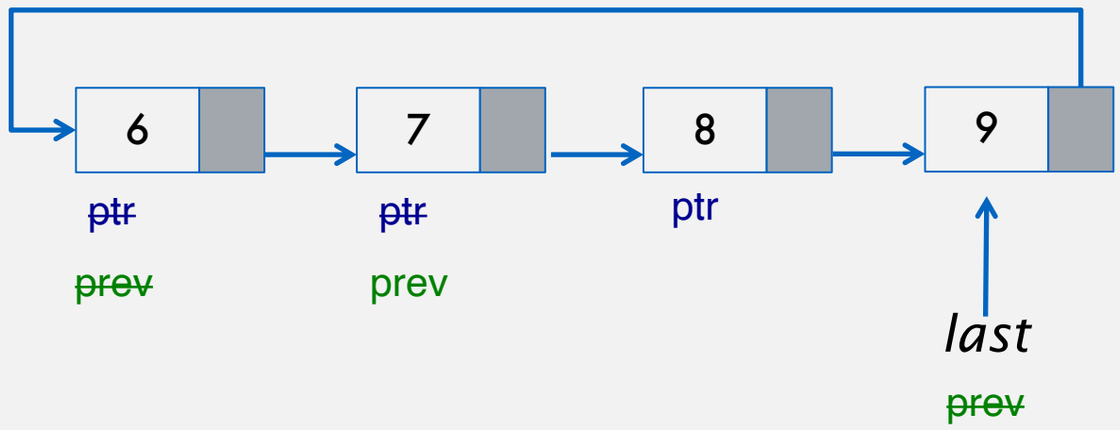
Worst: O(n)

Best: O(1)

Implementation of a circular Linked List

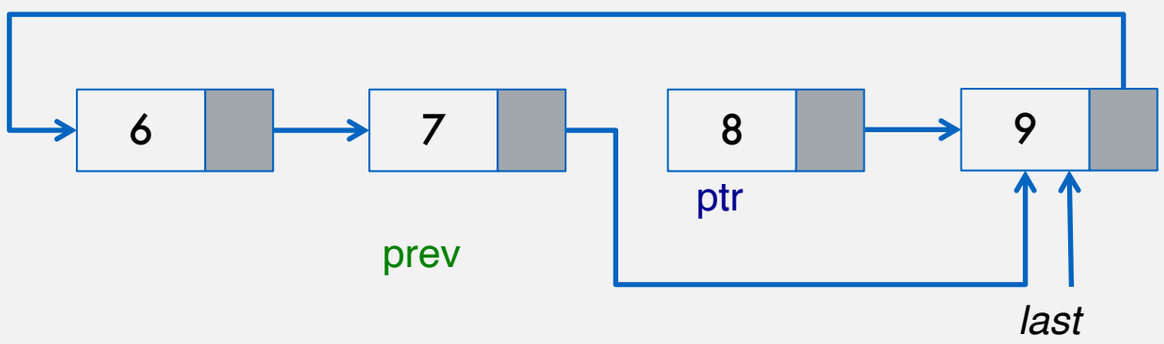
Delete target: removes the node with the target value

1. Find target




```
Node ptr = last.next;
Node prev = last;
do{
   if (ptr.data == target) {
    break;
   }
   prev = ptr;
   ptr = ptr.next;
} while (ptr != last.next);
```

2. Delete target




```
prev.next = ptr.next;
```

What is the running time?
- Worst if removing the last node: O(n)
- Best if removing the first node: O(1)

Three cases:
1. target not found (ptr == last.next)
2. target is last int the list (ptr == last)
3. target is found and is not the last in the list

Applications of Circular Linked Lists

LO 3.4, 3.5

- A circular linked list can be used to implement a queue, where only one reference is needed to implement enqueue or deque operations

- A circular linked list can be used to allocate a shared resource to multiple applications where each application is given some fixed amount of time and the operations can be continued until all applications complete their tasks

- A circular linked list can be used in a multi player game, where each player is given a chance in sequence, and first player is given a chance again in the second round once the last player is done in the first round.

Advantages and Disadvantages of Circular Linked Lists

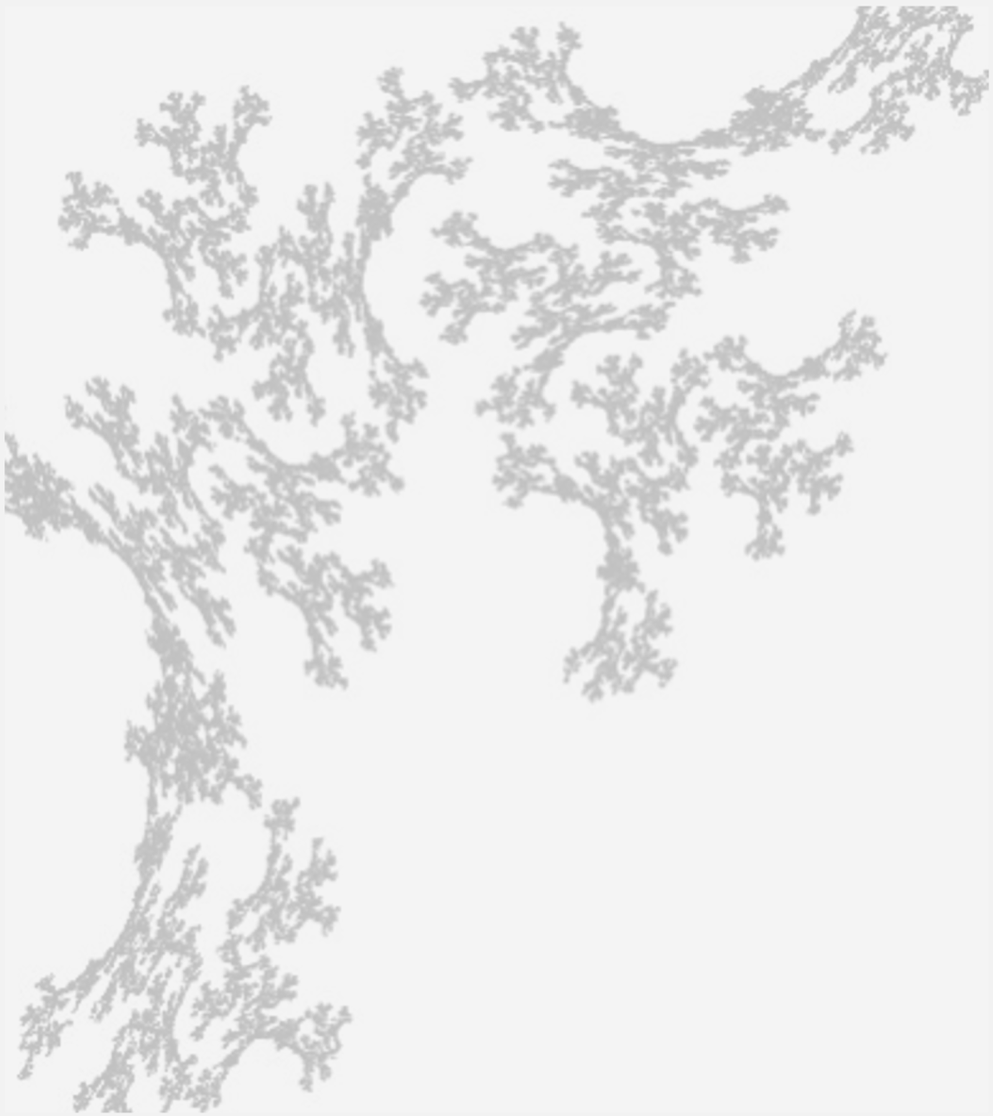
LO 3.4

Disadvantages

- Need to maintain a reference to the last node at all times
- It may be bit trickier to implement some operations such as addLast or addFirst

Advantages

- This is a great data structure to implement an application such as a multi-player game where players can be maintained in a circular linked list.
- Circular doubly linked list are used for implementation of practical data structures. (later)



Special Linked Structures

- *Introduction*
- *Linked Lists*
- *Doubly Linked Lists*
- *Circular Linked Lists*

http://ds.cs.Rutgers.edu

Credits: Some slides Adopted and modified from Sedgewick and Wayne

Last updated on 1/11/21 12:10 PM