CS151 Intro to Data Structures

LinkedLists

Announcements

- HW00 and Lab1 were due last Friday
- HW01 ExpandableArray Released - Due Friday Sep 19th
  - Builds on HW0 and Lab2
  - Lab2 also due Sep 19th

Outline

- Review - ExpandableArray
- Nested Classes
- **LinkedLists**

ExpandableArray

- Sequential, contiguous, memory layout
- Order from least to most expensive
  - insert at beginning
  - insert at end
  - remove from end
- Computational complexity:
  - Accessing an element?
    - O(1)
  - Inserting an element?
    - O(n)
  - Removing an element?
    - O(n)

Java.util.ArrayList

- https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

- `import java.util.ArrayList`

- `ExpandableArray` is a simple ArrayList

Methods of an ArrayList

`add(o)`	appends o at the end of list
`add(index, o)`	inserting given o at index, shifting list to the right
`get(index)`	returns the object found at index
`remove(index)`	removes the object found at index and returns it, shifting list to the left
`set(index, o)`	replaces object at given index with o
`size()`	returns the number of elements in list
`indexOf(o)`	returns the first index where o is found, or -1
`lastIndexOf(o)`	returns the last index where o is found, or -1
`clear()`	removes all

Nested Classes

- A class defined inside the definition of another class

- Benefits:
  - Encapsulation (data hiding and access control)

Nested Classes

- An instance of the inner class can't be created without an instance of the outer class.

- **Code**

Nested Classes - Access modifiers

- An inner class can access **all** members of the outer class

```
Person.this.name;
```

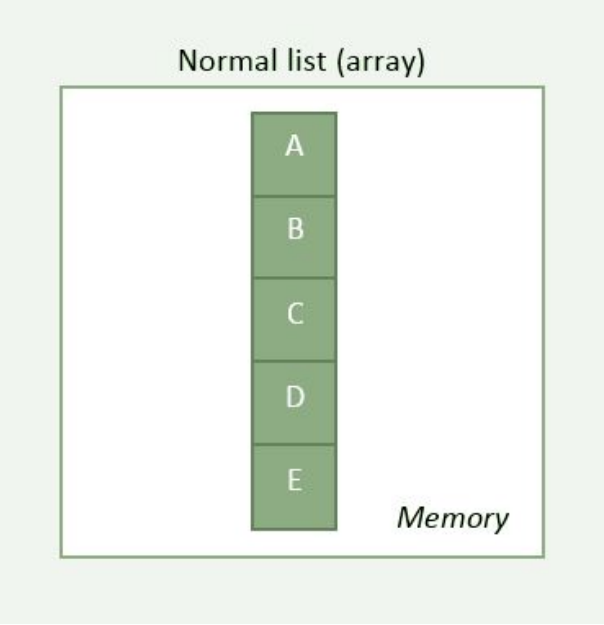
- An outer class can access **all** members in the inner class

- Even when they’re private!

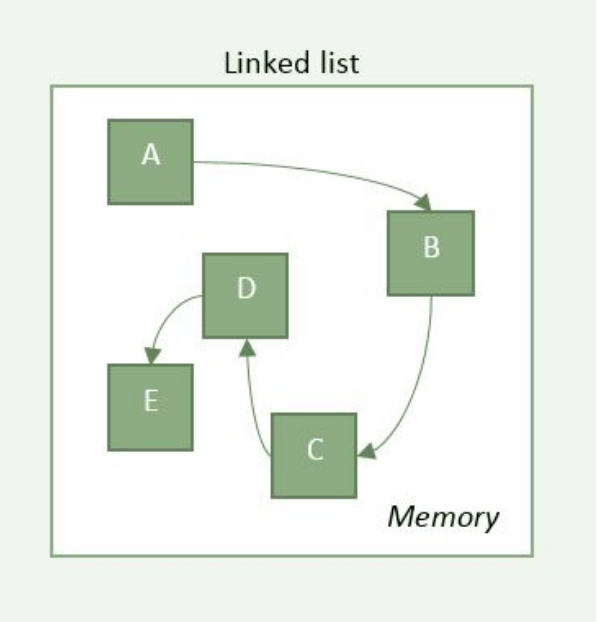
Linked List

List versus Array - memory

An array is a single consecutive piece of memory



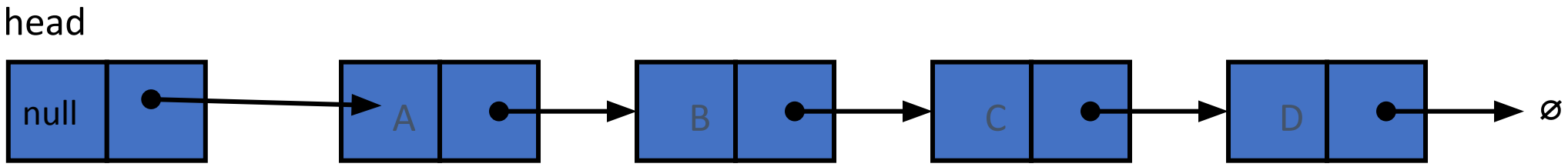
A list can be made of many disjoint pieces



Linked List

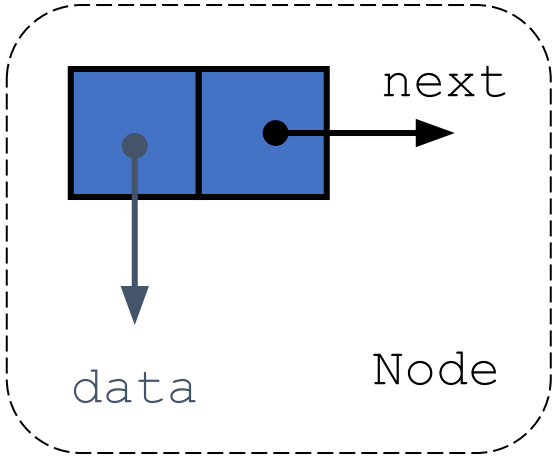
- A linked list is a lists of objects (**nodes**)
- The **nodes** form a linear sequence
- Linked lists are typically unbounded, that is, they can grow infinitely.

node: basic unit that contains data and one or more references or *links* to other nodes.



A node

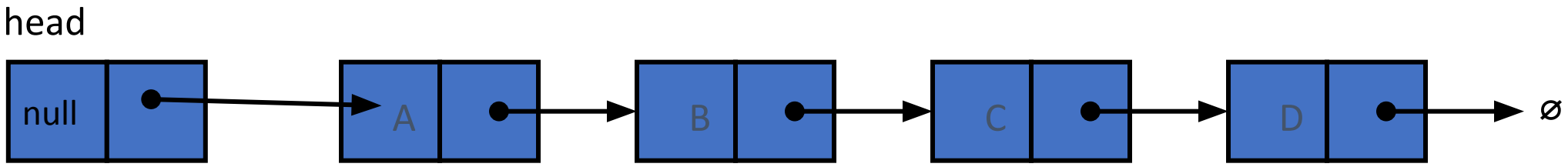
```
public class Node<T> {
   private T data;
   private Node next;
}
```



Linked List

How might we loop over all of the elements of a linked list?

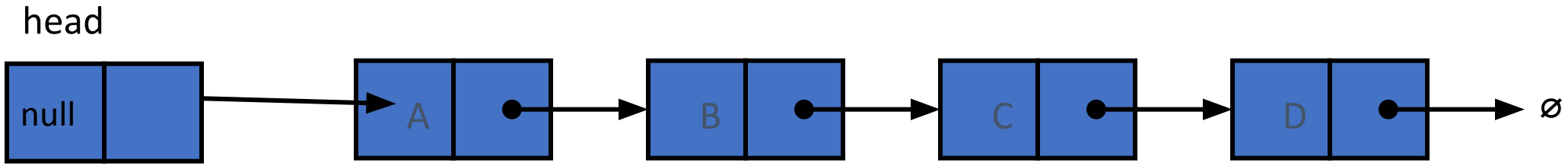
```
public class Node<T> {
  private T data;
  private Node next;
}
```



Linked List Operations

- Access
- Insertion
- Removal

Access Operation



- Check if the head node is what you are looking for
- Iterate through nodes:
  - Stop when found
  - Otherwise return null

Access Operation

Let’s code it

- Computational Complexity?
  - O(n)

Insert Operation

Let's code it

- Computational complexity?
  - Insert at head?
    - O(1)
  - Insert at tail?
    - O(n)
  - Insert at arbitrary location? (middle of list)
    - O(n)

Insert Operation

What if we keep a pointer to the tail?

```
private Node tail;
```

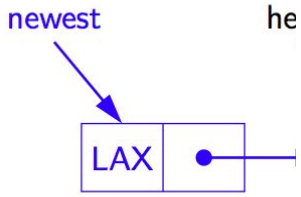
How does this change our insertTail method?

Computational complexity?

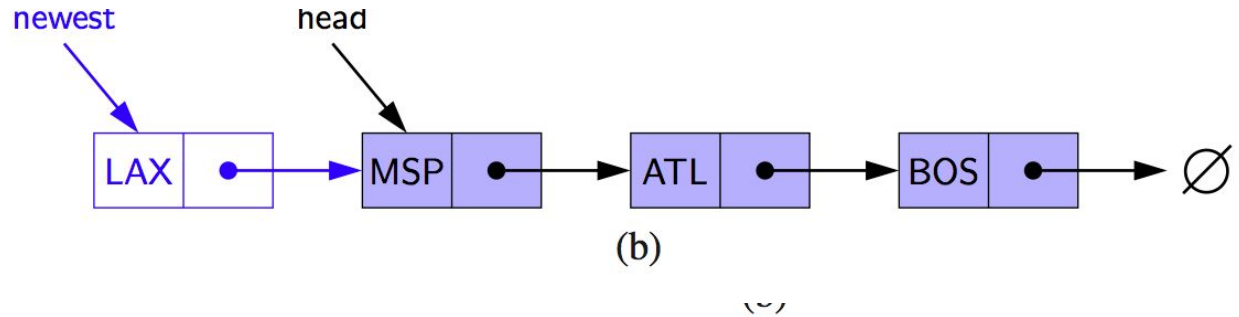
O(1)

Inserting at the Head

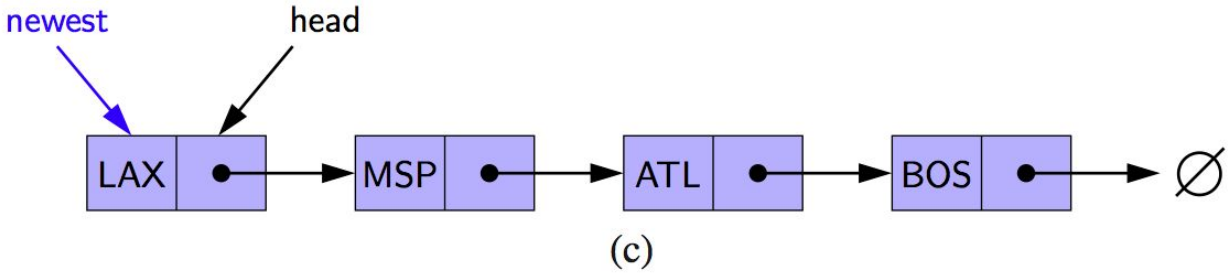
1. create a new node

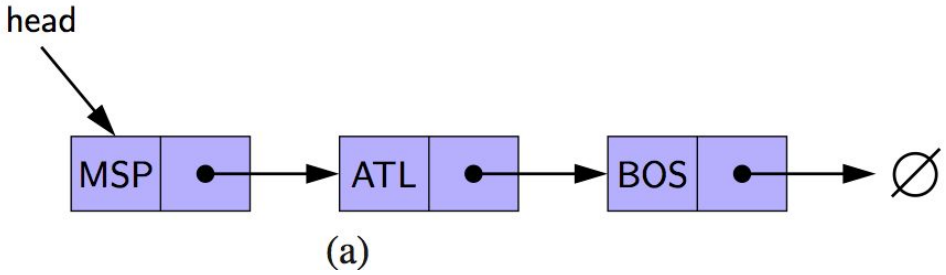


1. have new node point to old head



1. update head to point to new node

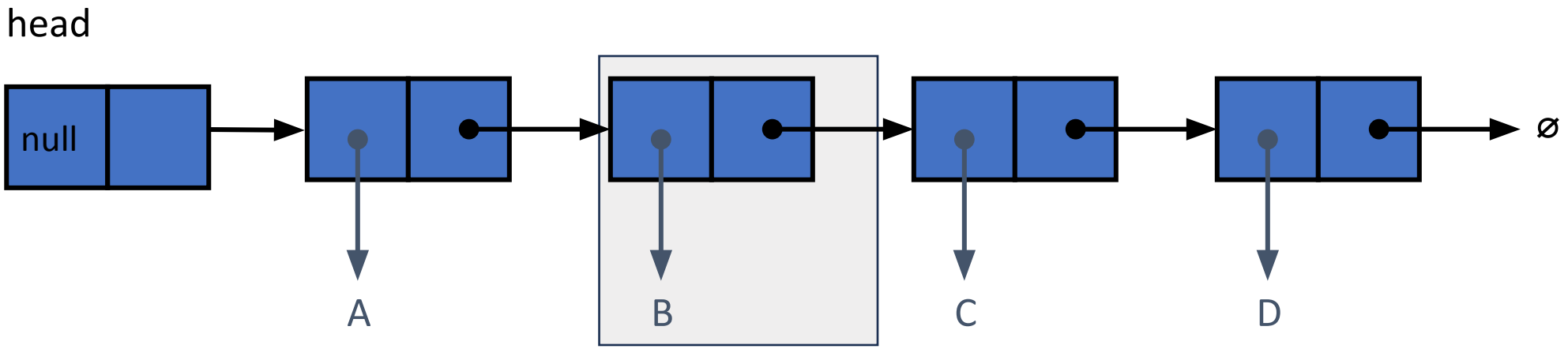


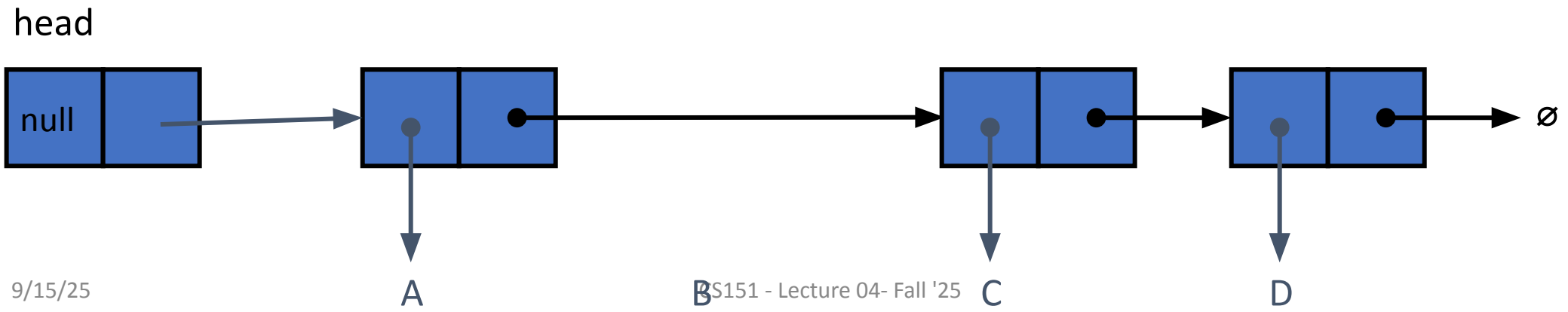


Remove Operation

- Let’s write it on the board quickly

Remove Operation `remove("B")`





Properties in LinkedList

What do we need to keep track of?

- Head
- Number of elements (optional)

Quiz

Rank from most efficient to least efficient:

- LinkedList find
- ExpandableArray find
- LinkedList insert at beginning
- ExpandableArray insert at beginning

Summary

- Linked Lists are data structures with disjoint memory
- not fixed size! Grows as elements are added
- O(n) access
- Insert at beginning is fast O(1)
- General insert is slow
  - in the worst case O(n)
- Removal is also slow O(n)

instanceof

- An operator that tests to see if an object is an instance of a specified type
- Every subclass object is an instance of its super class – not true the other way

```
class A {} class B extends A{} class C extends B{}
A[] as = {new A(), new B(), new C()};
for (int i=0; i<as.length; i++) {
  System.out.print((as[i] instanceof A)+ " ");
  System.out.print((as[i] instanceof B)+ " ");
  System.out.println(as[i] instanceof C);
}
```