

# CS151 Intro to Data Structures

## LinkedLists

# Announcements

- HW01 and Lab 2 due Wednesday Feb 4th
- Professor office hours today 3-5pm
  - Park 259

# Outline

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

# Runtime Complexity Review

- Big-O: mathematical notation used to describe the performance or complexity of an algorithm.
- Hardware independent
- Represents the upper bound of the time complexity in the worst-case scenario.
- Helps us understand how the runtime of an algorithm grows ***as the input size increases.***

# Constant Time operations

What are some operations that are independent of the input size?

- Assignments
- Declarations
- Accessing an index in an array
- `arr.length;`
- `mul`, `divide`, `sub`, `add`, `mod`, etc
- `greater than`, `less than`, etc
- `printing`
- `if (x > 100)`

# Example 1:

```
int n = Integer.parseInt(args[0]);
int sum = 0;
int i = 0;

while (i < n) {
    sum = sum + i;
    i++;
}

System.out.println(sum);
```

How does the runtime grow as a function of the input size?

Linearly!

$O(n)$

## Example 2:

```
int n = Integer.parseInt(args[0]);
int tot = 0;
int sum = 0;
int i = 0;

while (i < n) {
    tot = tot * i;
    i++;
}

i=0;
while (i < n) {
    sum = sum + i;
    i++;
}
```

How does the runtime grow as a function of the input size?

Linearly!

$O(n + n) = O(2n) = O(n)$

We care about the asymptotic case... The  $n$  factor dominates.

# Example 3:

```
int n = Integer.parseInt(args[0]);  
  
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        System.out.println(i, j);  
    }  
}
```

How does the runtime grow as a function of the input size?

Quadratically!

$O(n^2)$

We do  $n$  operations  $n$  times

# Example 4

```
int[] lst =  
    {1,2,3,5,7,12,19,34,55,67,99,101};  
  
int n = lst.length;  
int mid = floor(n/2);  
System.out.println(lst[mid]);
```

How does the runtime grow as a function of the size of `lst`?

Constant! The runtime is not affected by the number of elements in `lst`

$O(1)$

# Example 5:

```
int n = Integer.parseInt(args[0]);  
while (n > 1) {  
    println(n);  
    n = n/2;  
}
```

How does the runtime grow as a function of the size of  $n$ ?

$O(\log n)$

# Generics Review

- A way to write classes or methods that can operate on a variety of data types without being locked into specific types at the time of definition
- Write definitions with type parameters

```
public <T> void print(T x) {  
    System.out.println(x);  
}
```

# Generics Review

Generic classes:

```
public class GenericExpandableArray <T> {
```

T can be used as a type within the class

# Arrays Review

- Java arrays: Sequential, contiguous, memory layout
  - gives us  $O(1)$  access (if we know the index)
- fixed size
- `insert(E elem, int idx)`
- `remove(int idx)`
- `set(E elem, int idx)`

# ExpandableArray

- 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

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

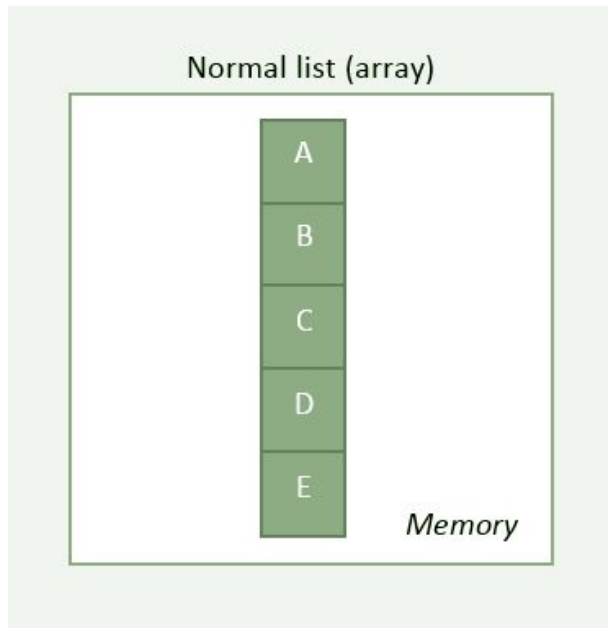
```
Person.this.name;
```

- An outer class can access **all** members in the inner class
- Even when they're private!

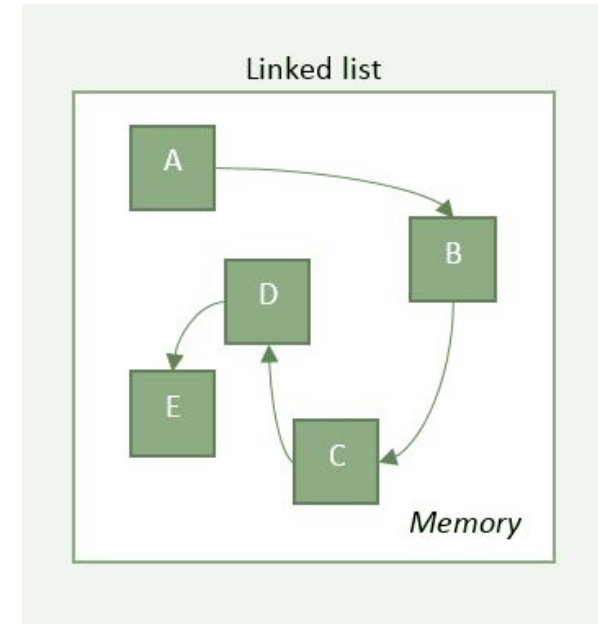
# Linked List

# List versus Array - memory

An array is a single consecutive piece of memory



A list can be made of many disjoint pieces

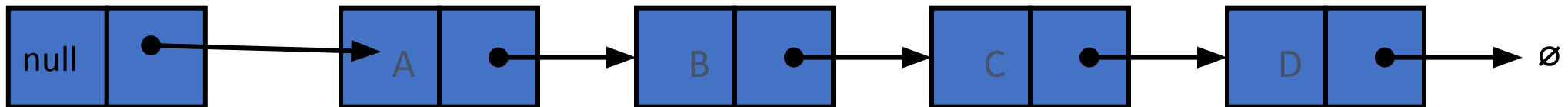


# Linked List

- 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.

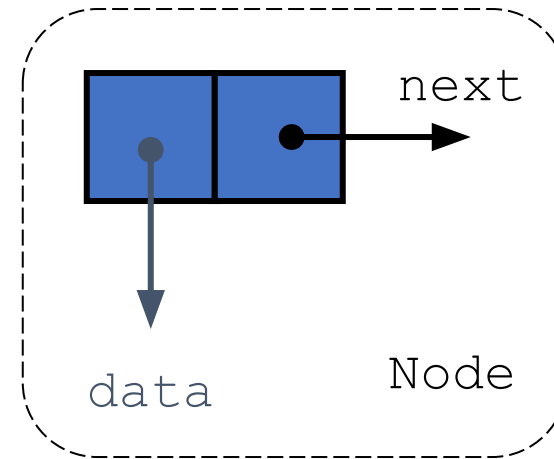
**node: basic unit that contains data and one or more references or *links* to other nodes.**

head



# A node

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

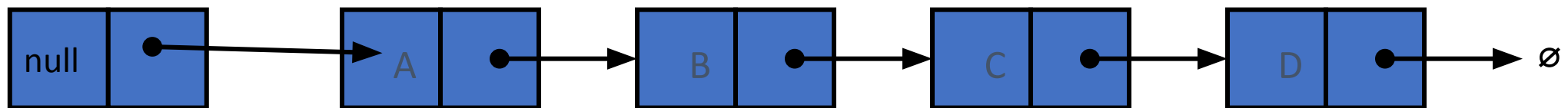


# Linked List

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

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

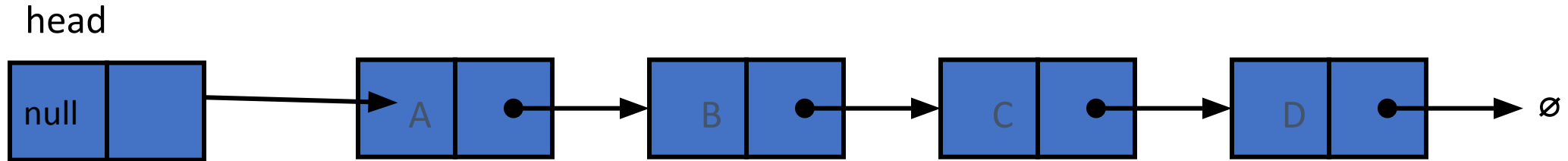
head



# 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;
```

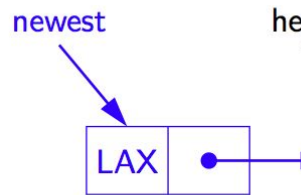
How does this change our insertTail method?

Computational complexity?

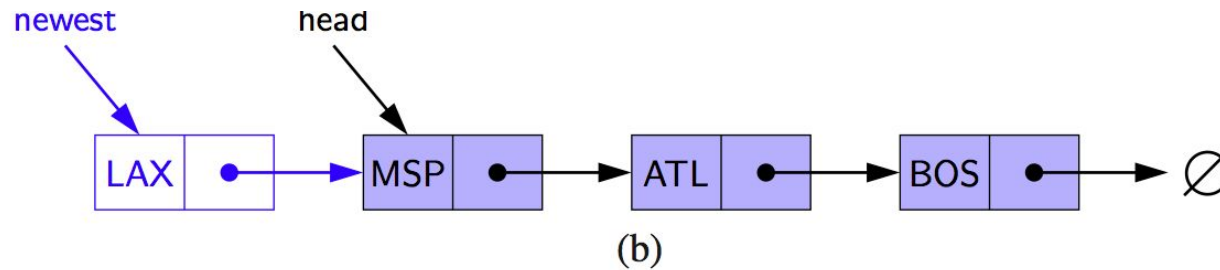
$O(1)$

# Inserting at the Head

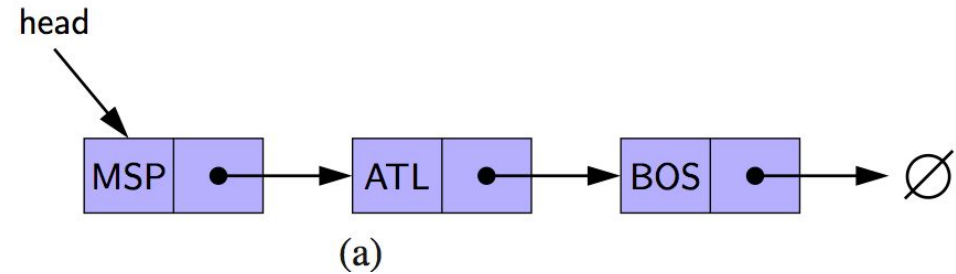
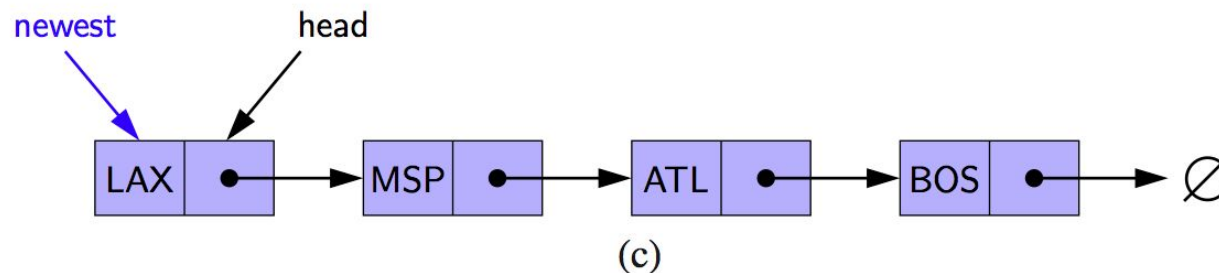
1. create a new node



1. have new node point to old head



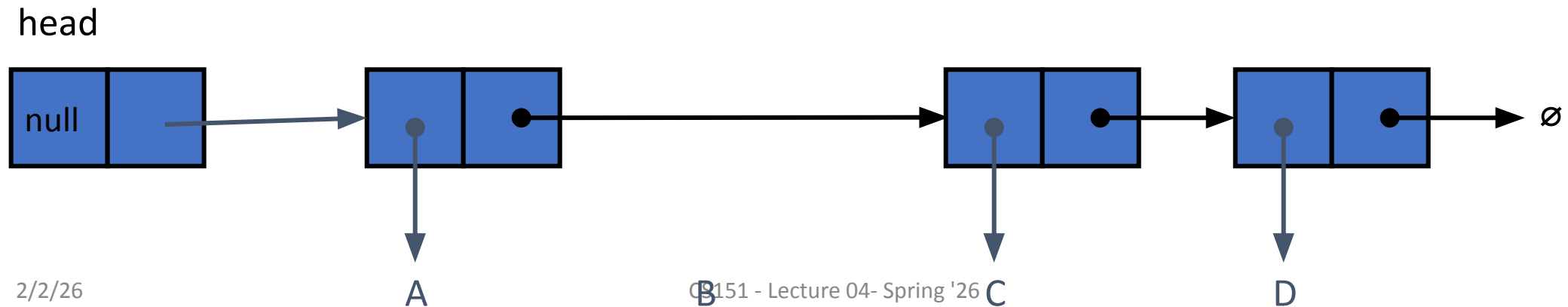
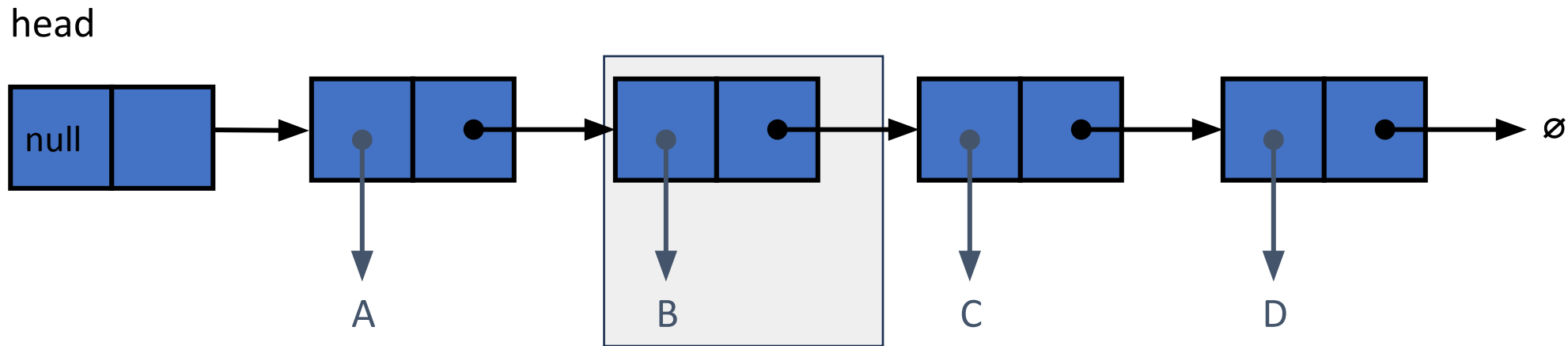
1. update head to point to new node



# 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);  
}
```