

CSE 250

Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 08: Sequences

Announcements

- PA1 Testing due Sunday at midnight
 - Be aware that course staff is not guaranteed to be available after 5PM or on weekends
 - Be thoughtful in your submissions to Autolab

Sequences (what are they?)

Fibonacci Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Characters in a String: 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'

Lines in a File

People in a queue

Sequences (what are they?)

Fibonacci Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Characters in a String: 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'

Lines in a File

People in a queue

An "ordered" collection of elements

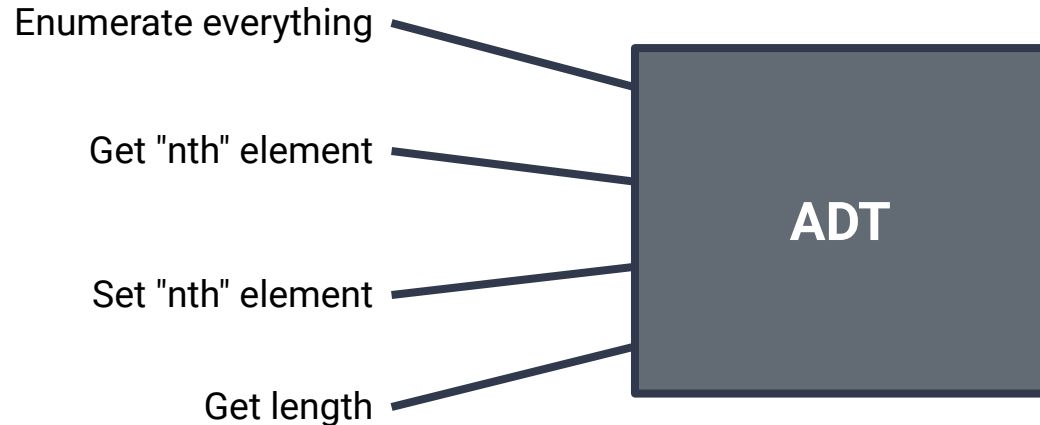
Sequences (what can you do with them?)

Sequences (what can you do with them?)

- Enumerate every element in sequence
 - ie: print out every element, sum every element
- Get the "nth" element
 - ie: what is the first element? what is the 42nd element?
- Modify the "nth" element
 - ie: set the first element to x, set the third element to y
- Count how many elements you have

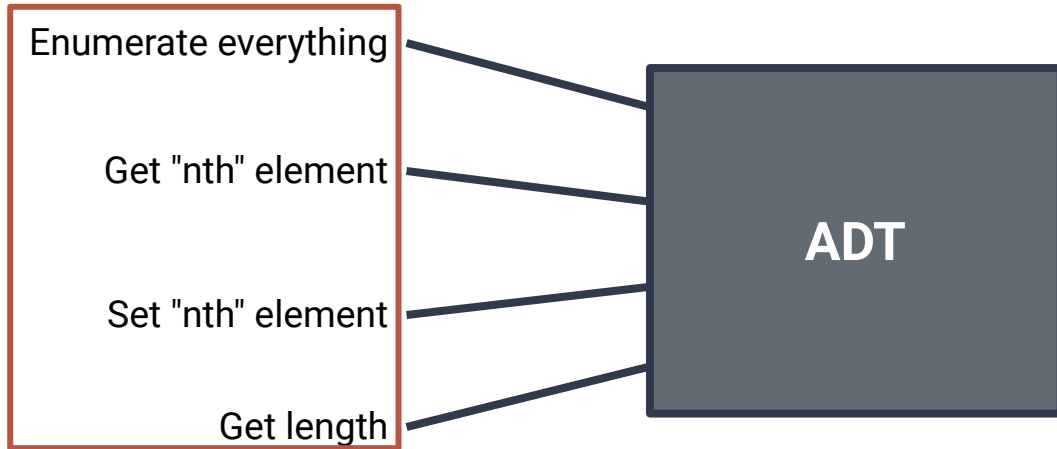
Abstract Data Types (ADTs)

The specification of **what** a data structure can do



Abstract Data Types (ADTs)

The specification of **what** a data structure can do



What's in the box? ...we don't know, and in some sense...we don't care

Usage is governed by **what** we can do, not **how** it is done

The Sequence ADT

T get(int idx)

Get the element (of type T) at position **idx**

T set(int idx, T value)

Set the element (of type T) at position **idx** to a new value

int length

Get the number of elements in the seq

Iterator<T> iterator()

Get access to view all elements in the sequence, in order, once

So...what's in the box?
(how do we implement it)

A Brief Aside on RAM (220 crossover)

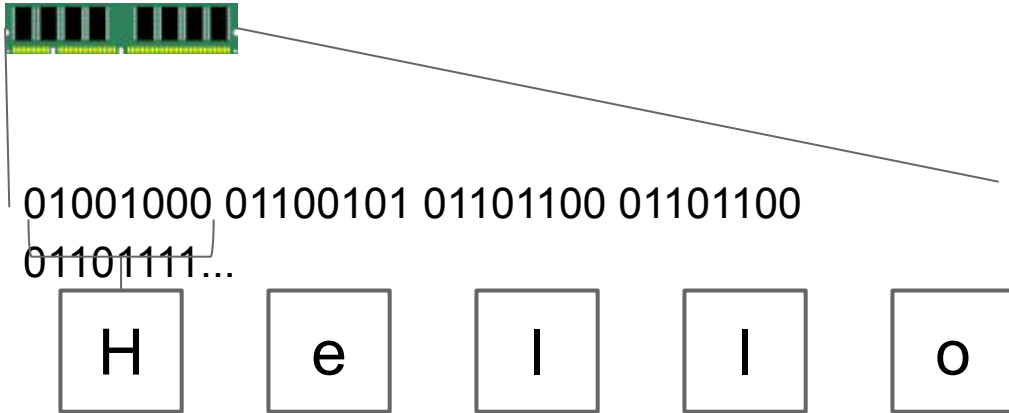


A Brief Aside on RAM (220 crossover)

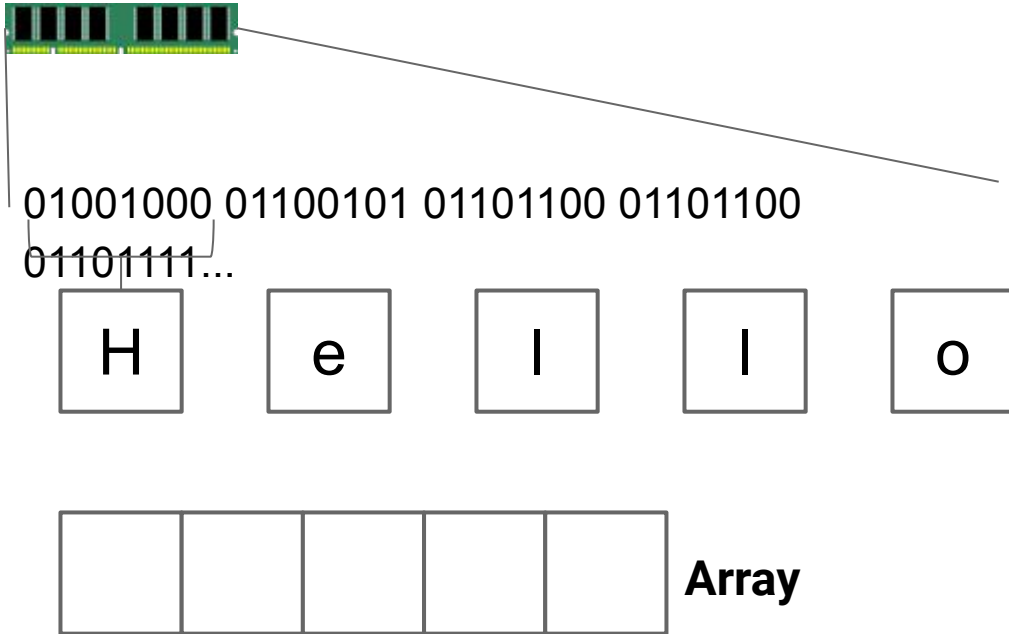


01001000 01100101 01101100 01101100
01101111...

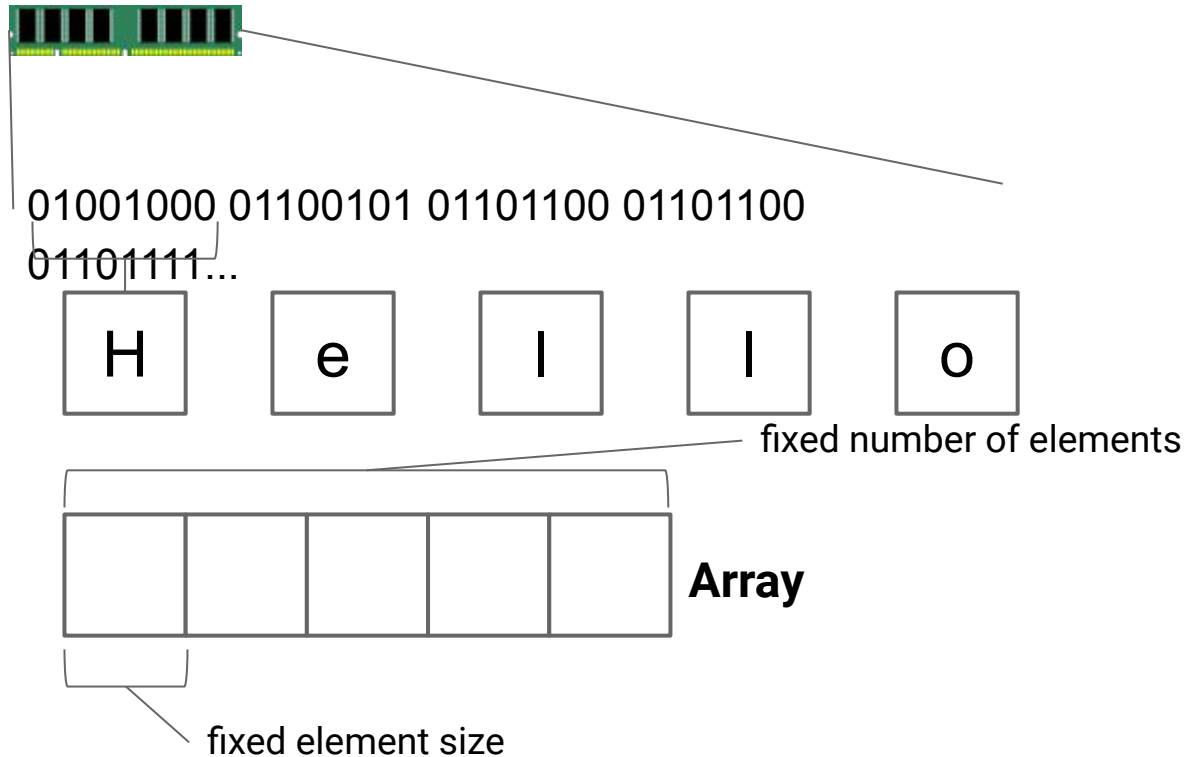
A Brief Aside on RAM (220 crossover)



A Brief Aside on RAM (220 crossover)



A Brief Aside on RAM (220 crossover)



RAM

Allocation with new T:

Go find some unused part of memory that is big enough to fit a T, mark it as used, and return the **address** of that location in memory.

RAM

Allocation with new T:

Go find some unused part of memory that is big enough to fit a T, mark it as used, and return the **address** of that location in memory.

```
1 int[] arr = new int[50];
```

The above code allocates $50 * 4 = 200$ bytes of memory*
(a single Java `int` takes of 4 bytes in memory)

* *slightly more actually...see next slide*

Arrays in Detail

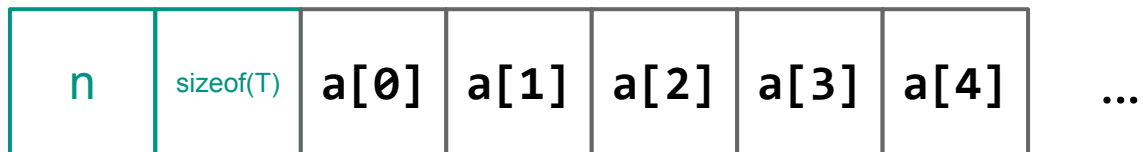
What does an array of n items of type T actually look like?

- 4 bytes for n (optional)
- 4 bytes for `sizeof(T)` (optional)
- $n * \text{sizeof}(T)$ bytes for the data

Arrays in Detail

What does an array of n items of type T actually look like?

- 4 bytes for n (optional)
- 4 bytes for `sizeof(T)` (optional)
- $n * \text{sizeof}(T)$ bytes for the data



Arrays in Detail

How would we implement the methods of the Sequence ADT for an Array:

```
T get(int idx)
```

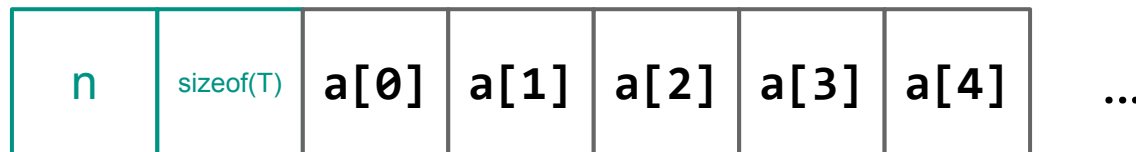
```
T set(int idx, T value)
```

```
int length
```

Arrays in Detail

What does an array of n items of type T actually look like?

- 4 bytes for n (optional)
- 4 bytes for `sizeof(T)` (optional)
- $n * \text{sizeof}(T)$ bytes for the data



The length is stored in the memory allocated for the array... $\Theta(1)$ time to access

Arrays in Detail

How would we implement the methods of the Sequence ADT for an Array:

`T get(int idx)`

`T set(int idx, T value)`

`int length`

Access the `length` field in constant time

$\Theta(1)$

Implementing get/set

```
1 int[] arr = new int[50];
```

If `arr` is at address `a`, where should you look for `arr[19]`?

Implementing get/set

```
1 int[] arr = new int[50];
```

If `arr` is at address `a`, where should you look for `arr[19]`?

- $a + 19 * 4$

Implementing get/set

```
1 int[] arr = new int[50];
```

If `arr` is at address `a`, where should you look for `arr[19]`?

- $a + 19 * 4$ (does this computation depend on the size of `arr`?)

What is the complexity?

Implementing get/set

```
1 int[] arr = new int[50];
```

If `arr` is at address `a`, where should you look for `arr[19]`?

- $a + 19 * 4$ (does this computation depend on the size of `arr`?)

What is the complexity? $\Theta(1)$

Implementing get/set

```
1 int[] arr = new int[50];
```

If `arr` is at address `a`, where should you look for `arr[19]`?

- $a + 19 * 4$ (does this computation depend on the size of `arr`?)

What is the complexity? $\Theta(1)$

What about `a[55]`?

Implementing get/set

```
1 int[] arr = new int[50];
```

If `arr` is at address `a`, where should you look for `arr[19]`?

- $a + 19 * 4$ (does this computation depend on the size of `arr`?)

What is the complexity? $\Theta(1)$

What about `a[55]`?

- $a + 55 * 4$...but that memory was not reserved for this array.
- Java will prevent you from accessing an *out of bounds* element

Arrays in Detail

How would we implement the methods of the Sequence ADT for an Array:

T get(int idx)

Compute the address of the element in constant time $\Theta(1)$

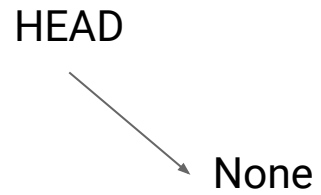
T set(int idx, T value)

Compute the address of the element in constant time $\Theta(1)$

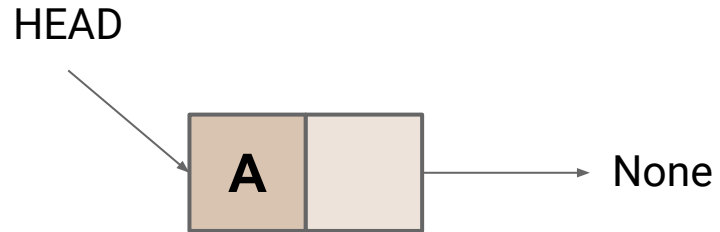
int length

Access the **length** field in constant time $\Theta(1)$

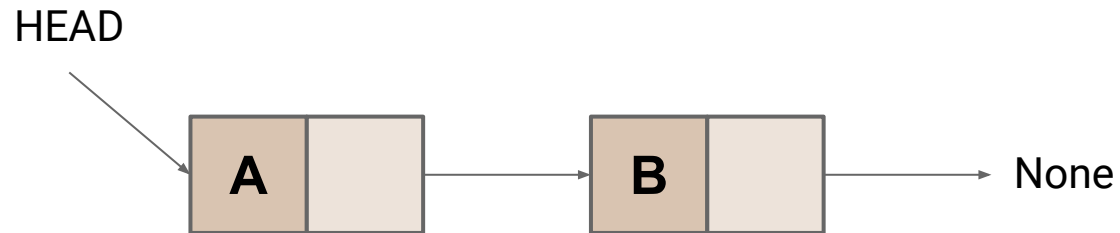
Linked Lists



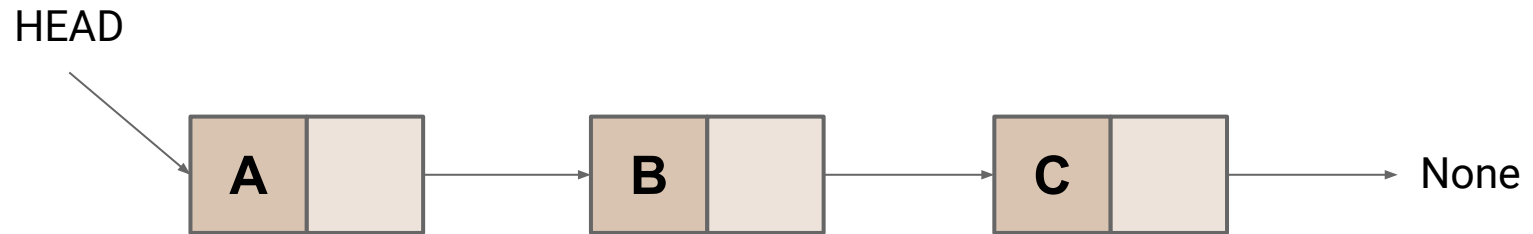
Linked Lists



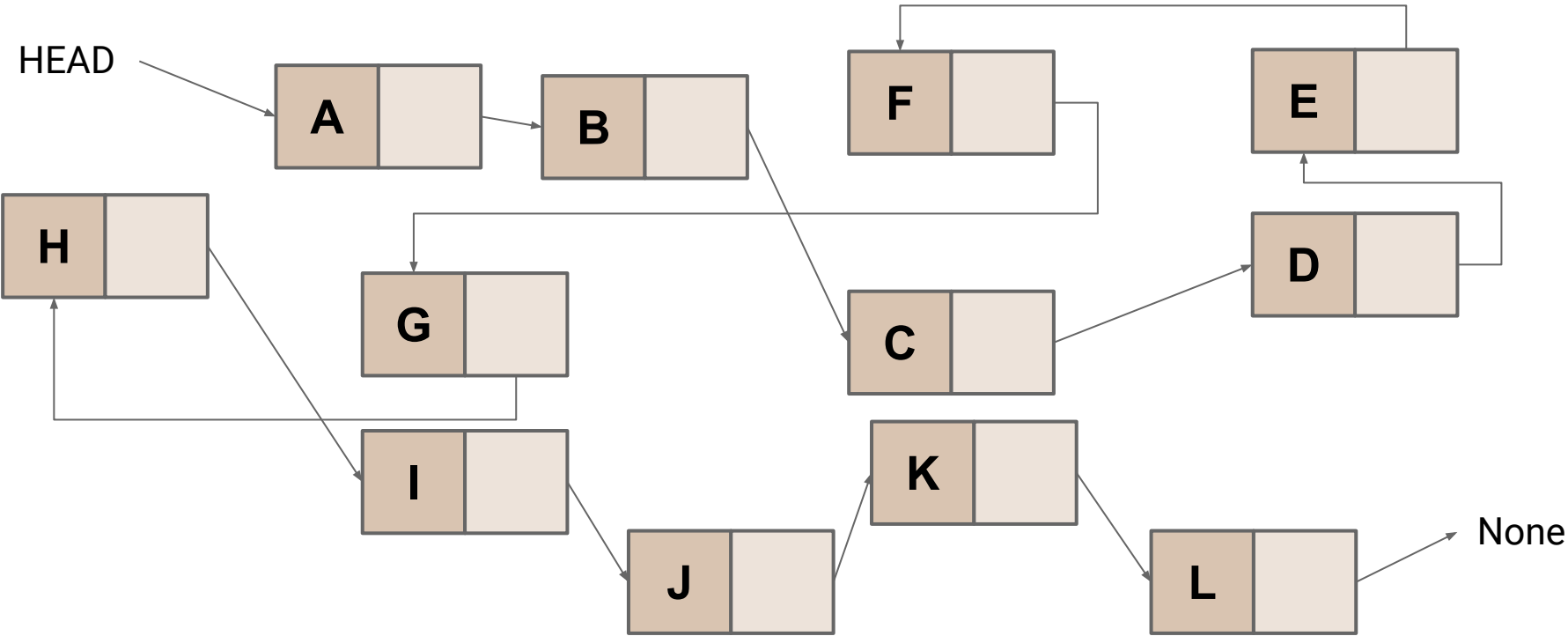
Linked Lists



Linked Lists



Linked Lists



Linked Lists in Detail

```
1 class LinkedList<T> {  
2     Optional<LinkedListNode<T>> head = Optional.empty();  
3     /* ... */  
4 }
```

Class for our list, which right now just has a **Optional** reference to **head**

```
1 class LinkedListNode<T> {  
2     T value;  
3     Optional<LinkedListNode<T>> next = Optional.empty();  
4 }
```

Class for a node in the list, which has a **value**, and an **Optional** reference to the **next** node

Linked Lists in Detail

```
1 class LinkedList<T> {  
2     Optional<LinkedListNode<T>> head = Optional.empty();  
3     /* ... */  
4 }
```

Class for our list, which right now just has a **Optional** reference to **head**

```
1 class LinkedListNode<T> {  
2     T value;  
3     Optional<LinkedListNode<T>> next = Optional.empty();  
4 }
```

Class for a node in the list, which has a **value**, and an **Optional** reference to the **next** node

What is Optional<T>...a brief digression

- Let's say we have a function that we know can possibly return `null`
- What can go wrong in the following code snippet?

```
1 Integer x = functionThatCanReturnNull();  
2 x.doAThing();
```

What is Optional<T>...a brief digression

- Let's say we have a function that we know can possibly return `null`
- What can go wrong in the following code snippet?

```
1 Integer x = functionThatCanReturnNull();  
2 x.doAThing();
```

`java.lang.NullPointerException` (runtime error)

What is Optional<T>...a brief digression

```
1 Integer x = functionThatCanReturnNull();  
2 if (x == null) { /* do something special */ }  
3 else { x.doAThing(); }
```

We need to add a check for `null` to avoid this...but this is easy to forget

What if our function returns `Optional<Integer>` instead?

What is `Optional<T>`...a brief digression

- Now our function returns `Optional<Integer>`
- What can go wrong in the following code snippet?

```
1 Optional<Integer> x = functionThatCanReturnEmpty();  
2 x.doAThing();
```


What is Optional<T>...a brief digression

- Now our function returns `Optional<Integer>`
- What can go wrong in the following code snippet?

```
1 Optional<Integer> x = functionThatCanReturnEmpty();  
2 x.doAThing();
```

`Cannot resolve method doAThing() in Optional`
(compile error)

What is Optional<T>...a brief digression

```
1 Optional<Integer> x = functionThatCanReturnNull();  
2 if (x.isPresent()) { x.get().doAThing(); }  
3 else { /* do something special */ }
```

Java makes us do something sensible!

What is Option[T]...a brief digression

Creating Optional objects:

```
Optional.empty()           // Like null
Optional.of(x)             // Optional object w with value x
Optional.ofNullable(x)    // If x is null same as .empty()
```

Using Optional objects:

```
.isPresent()              // True if there is a value
.get()                    // gets the value
.orElse(y)                // return value if present, y if not
```

Linked Lists in Detail

How do we implement the methods of the Sequence ADT for a Linked List:

```
T get(int idx)
```

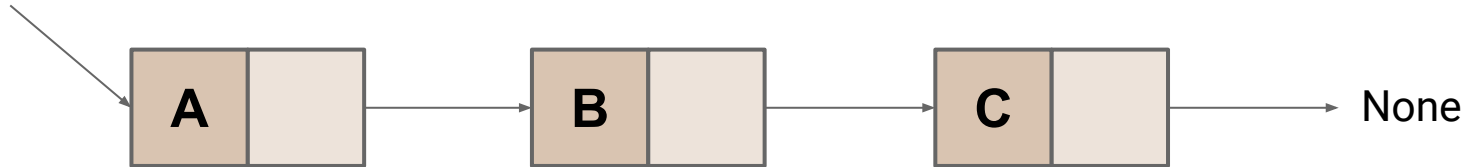
```
T set(int idx, T value)
```

```
int length
```

Implementing get/set

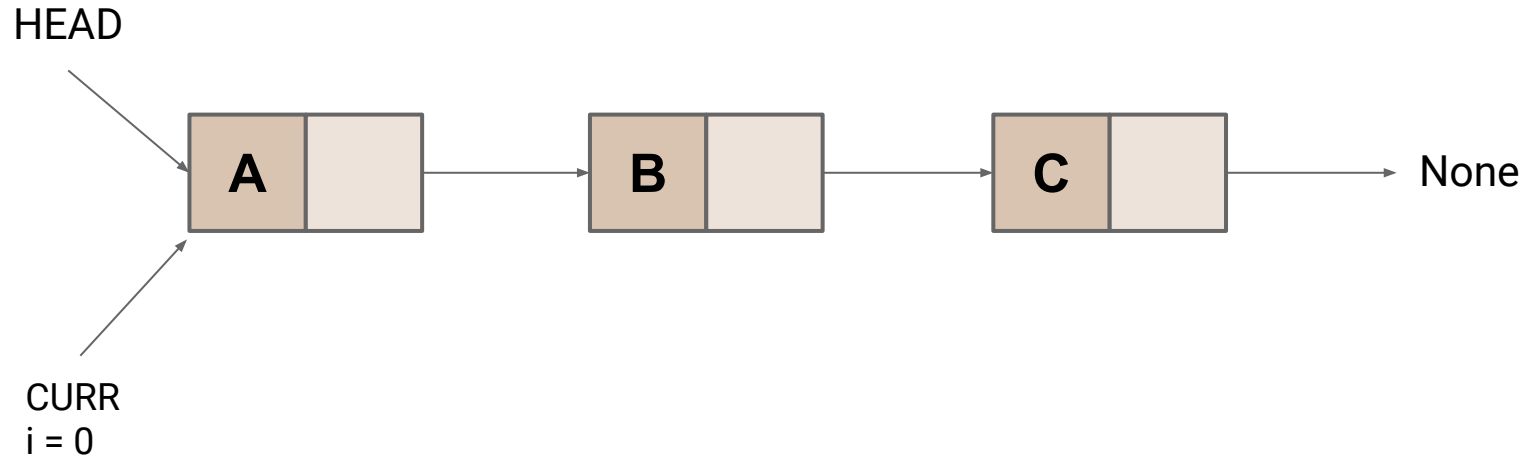
get(2)

HEAD



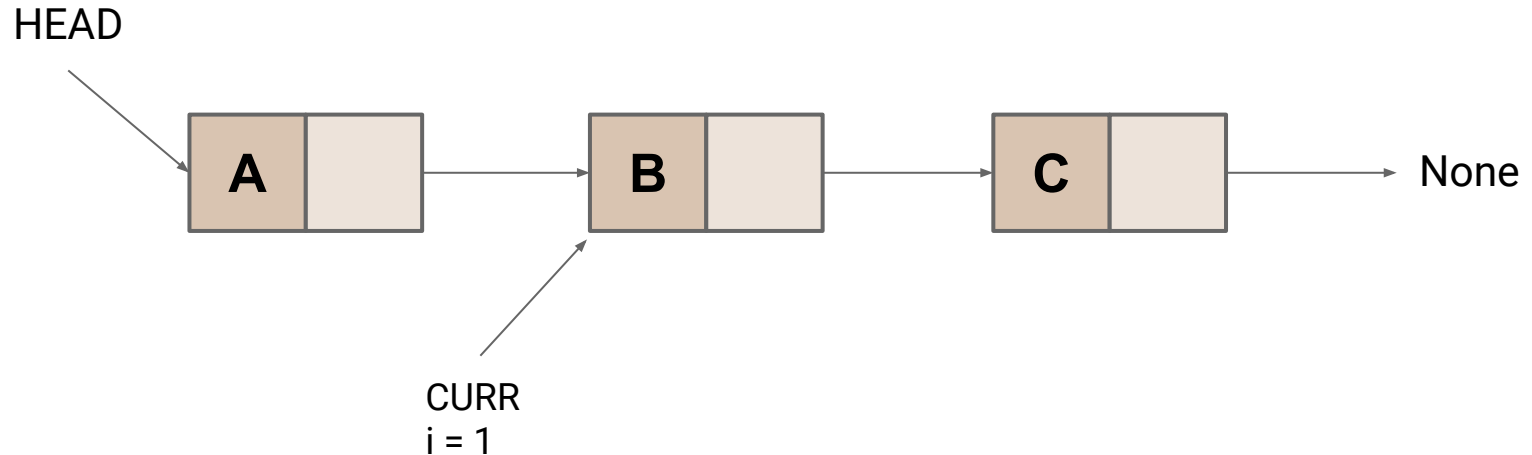
Implementing get/set

get(2)



Implementing get/set

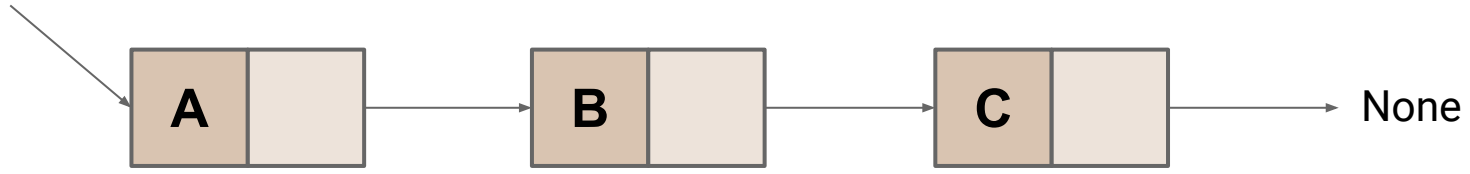
get(2)



Implementing get/set

get(2)

HEAD



Implementing get/set

```
1 public T get(int idx) {
2     int i = 0;
3     Optional<LinkedListNode<T>> curr = head;
4     while(i < idx) {
5         if (!curr.isPresent()) { throw new IndexOutOfBoundsException(); }
6         i++;
7         curr = curr.get().next;
8     }
9     if(!curr.isPresent()) { throw new IndexOutOfBoundsException(); }
10    return curr.get().value;
11 }
```

Implementing get/set

```
1 public T get(int idx) {  
2     int i = 0;  
3     Optional<LinkedListNode<T>> curr = head;  
4     while(i < idx) {  
5         if (!curr.isPresent()) { throw new IndexOutOfBoundsException(); }  
6         i++;  
7         curr = curr.get().next;  
8     }  
9     if(!curr.isPresent()) { throw new IndexOutOfBoundsException(); }  
10    return curr.get().value;  
11 }
```

All of this is $\Theta(1)$

Implementing get/set

```
1 public T get(int idx) {  
2      $\Theta(1)$   
3     while(i < idx) {  
4          $\Theta(1)$   
5     }  
6      $\Theta(1)$   
7 }
```

Implementing get/set

```
1 public T get(int idx) {  
2      $\Theta(1)$   
3      $\Theta(\text{idx})$   
4      $\Theta(1)$   
5 }
```

Complexity: $\Theta(\text{idx}) \subset O(n)$

Linked Lists in Detail

How do we implement the methods of the Sequence ADT for a Linked List:

T get(int idx)

Go node-by-node until you reach **idx** $\Theta(\text{idx}) \subset O(n)$

T set(int idx, T value)

Go node-by-node until you reach **idx** $\Theta(\text{idx}) \subset O(n)$

int length

Implementing length

```
1 public int length() {  
2     int i = 0;  
3     Optional<LinkedListNode<T>> curr = head;  
4     while(curr.isPresent()) { i++; curr = curr.get().next; }  
5     return i;  
6 }
```

Implementing length

```
1 public int length() {  
2      $\Theta(1)$   
3     while(curr.isPresent()) {  $\Theta(1)$  }  
4      $\Theta(1)$   
5 }
```

Implementing length

```
1 public int length() {  
2      $\Theta(1)$   
3      $\Theta(n)$   
4      $\Theta(1)$   
5 }
```

Complexity: $\Theta(n)$
Can we do better?

Implementing Length

Idea: Have the Linked List class store the length

```
1 class LinkedList<T> {  
2     Optional<LinkedListNode<T>> head = Optional.empty();  
3     int length; ←  
4     /* ... */  
5 }
```

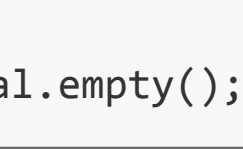


Now complexity of getting **length** is $\Theta(1)$

Implementing Length

Idea: Have the Linked List class store the length

```
1 class LinkedList<T> {  
2     Optional<LinkedListNode<T>> head = Optional.empty();  
3     int length; ←  
4     /* ... */  
5 }
```



Now complexity of getting **length** is $\Theta(1)$

How much extra space is required? $\Theta(1)$

How much extra work is required to insert/remove? $\Theta(1)$

Implementing length

Idea: Have the Linked List class store the length

```
1 class LinkedList<T> {  
2     0  
3     i  
4     /  
5 }
```

Common trade-off: Sometimes storing extra information can decrease complexity!

Now complexity of getting **length** is $\Theta(1)$

How much extra space is required? $\Theta(1)$

How much extra work is required to insert/remove? $\Theta(1)$

Access by-Reference vs by-Index

Complexity of getting the value of the n th node in a Linked List?

Complexity of getting the value of the n th node if we have a reference to that node?

Complexity of getting the value of $(n+1)$ th node if we have a reference to the n th node?

Complexity of getting the value of $(n-1)$ th node if we have a reference to the n th node?

Access by-Reference vs by-Index

Complexity of getting the value of the n th node in a Linked List? $\Theta(n)$

Complexity of getting the value of the n th node if we have a reference to that node? $\Theta(1)$

Complexity of getting the value of $(n+1)$ th node if we have a reference to the n th node? $\Theta(1)$

Complexity of getting the value of $(n-1)$ th node if we have a reference to the n th node? $\Theta(n)$

Doubly Linked Lists

```
1 class LinkedList<T> {  
2     Optional<LinkedListNode<T>> head = Optional.empty();  
3     Optional<LinkedListNode<T>> tail = Optional.empty();  
4     int length;  
5 }
```

```
1 class LinkedListNode<T> {  
2     T value;  
3     Optional<LinkedListNode<T>> next = Optional.empty();  
4     Optional<LinkedListNode<T>> prev = Optional.empty();  
6 }
```