

Linked List

Recall - Array

- Sequential
 - One continuous block of memory
 - Random access based on memory address
 - $\text{address} = \text{first_address} + (\text{element_size} * \text{index})$
- Fixed Size
 - Since memory adjacent to the block may be used
 - Efficient when you know how many elements you'll need to store

Array

Program Stack	
Main Frame	name:myArray, value:1503

Program Heap	
1503	myArray[0]
...	myArray[1]
...	myArray[2]
...	myArray[3]
[used by another program]	

- Arrays are stored on the heap
- Pointer to index 0 goes on the stack
- add $\text{index} * \text{sizeofElement}$ to 1503 to find each element
- This is called random access

Recall - Linked List

- Sequential
 - Spread across memory
 - Each element knows the memory address of the next element
 - Follow the addresses to find each element
- Variable Size
 - Store new element anywhere in memory
 - Add store the memory address in the last element
 - or new element stores address of first element

Linked List

Program Stack	
Main Frame	name:myList, value:506

- myList stores a list containing: [5,3,1]
- Last link stores null
 - We say the list is "null terminated"
 - When we read a value of null we know we reached the end of the list

Program Heap	
506	name:value, value:5
...	name:next, value:795

Program Heap	
795	name:value, value:3
...	name:next, value:416

Program Heap	
416	name:value, value:1
...	name:next, value:299

Program Heap	
299	name:value, value:null
...	name:next, value:null

Linked List

Program Stack	
Main Frame	name:myList, value:506

Program Heap	
506	name:value, value:5
...	name:next, value:795

Program Heap	
795	name:value, value:3
...	name:next, value:416

Program Heap	
416	name:value, value:1
...	name:next, value:299

Program Heap	
299	name:value, value:null
...	name:next, value:null

```
class LinkedListNode[A](var value: A, var next: LinkedListNode[A]) {  
}
```

```
var tail: LinkedListNode[Int] = new LinkedListNode[Int](null, null)  
  
var myList: LinkedListNode[Int] = new LinkedListNode[Int](1, tail)  
myList = new LinkedListNode[Int](3, myList)  
myList = new LinkedListNode[Int](5, myList)
```

- We create our own linked list class by defining a node
 - A node represents one "link" in the list
- The list itself is a reference to the first/head node

Lecture Question

Task: Write a prepend method for our linked list

- Write a method in the `datastructures.LinkedListNode` class (from the repo) named `prepend` that:
 - Takes a value of type `A` as its parameter
 - Prepends the input to the front of this list (assume the method is called on the head node)
 - Returns a reference to the new head of the list

* This question will be open until midnight