CS310
Data Structures

K. Raven Russell
krusselc@gmu.edu
George Mason University

Today

- **Last Lecture**
  - More Linked Lists
- **Today**
  - Stacks
  - Queues

Linked List Variants

- **Node Fields**
  - Reference to next node (“singly”)
  - Reference to previous and next node (“doubly”)
- **List Fields**
  - Keep reference to head node
  - Keep reference to tail node (optional)
    - *Big-O changes for some operations without this*
  - Track the size (optional)
    - *Big-O changes for some operations without this*

Warm-up Big-O for LIST

Operation Implementation	get set	add remove (end)	insert remove (start)	insert remove (middle)	search	grow? shrink?
Static Array List						No
Dynamic Array List						Yes
Singly Linked List						Yes
Doubly Linked List						Yes

Linked List Variants Comparison

- Remember: to insert and remove from the middle you first have to search for the correct position which is O(n)

Operation Implementation	get set	add remove (end)	insert remove (start)	insert remove (middle)	search	grow? shrink?
Singly Linked List	N	1, N	1	N	N	Yes
Doubly Linked List	N	1	1	N	N	Yes

- Singly linked list add is constant time but remove requires searching down to node before last to set to null.
- Doubly linked uses more memory, but still O(n)

Which Implementation is Best?

- **Array / Static Array** - “row” of memory
  - can run out of space
- **Dynamic Arrays** - arrays that can grow
  - cost to copy repeatedly (not so bad)
  - insert/remove expensive (not good at all - expensive)
- **Linked Lists** - tiny blocks of memory “linked” together
  - no “quick” memory access
  - extra memory to represent compared to array
  - fewer “expensive” memory moves

List Implementation Summary

– Though arrays are limited in functionality, constants for arrays are much faster

Operation Implementation	get set	add remove (end)	insert remove (start)	insert remove (middle)	search	grow? shrink?
Array	1	-	-	-	-	No
Static Array List	1	1	N	N	N	No
Dynamic Array List	1	1*	N	N	N	Yes
Singly Linked List	N	1,N	1	N	N	Yes
Doubly Linked List	N	1	1	N	N	Yes
**	1	-	-	-	1	Yes

* Amortized analysis

** Hash Tables, will cover later this semester

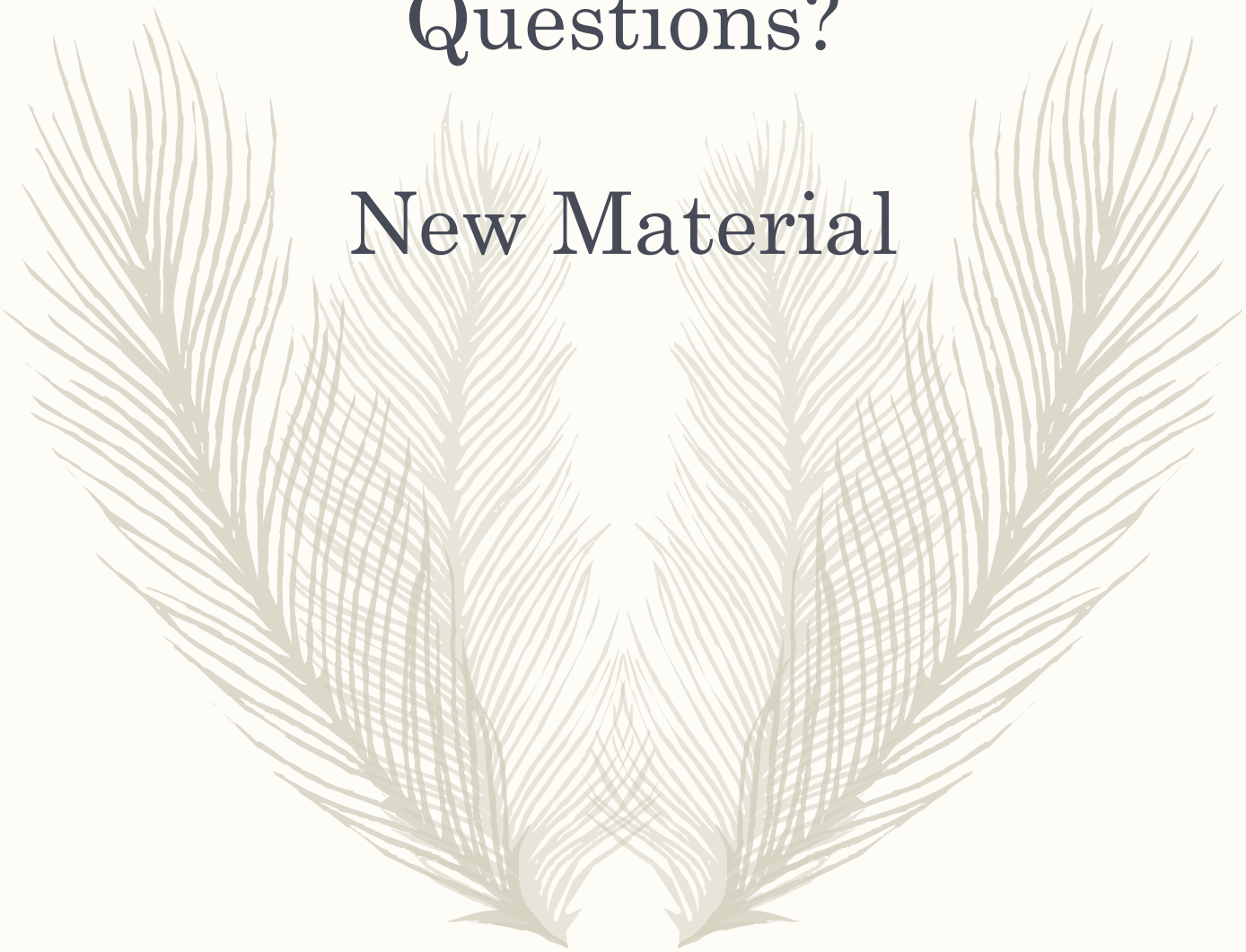
General Rule in Data Structures

- **Arrays are simple**
  - get/set anything
  - add/remove is obvious (need size variable)
  - very clear how data is laid out

- Just about every other data structure is less so
  - get/set nontrivial
  - must preserve some internal structure - control access
  - element-by-element access takes work (time)

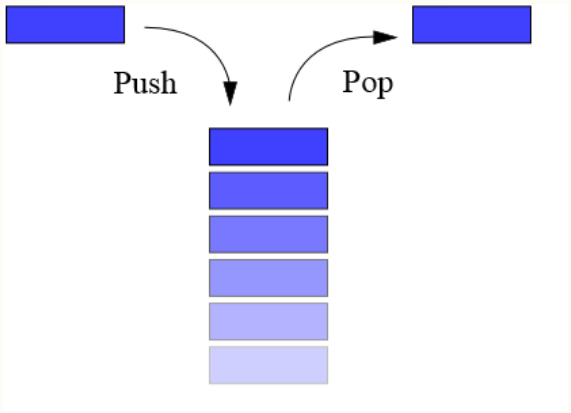
Questions?

New Material



Stacks and Queues

- Stack
  - Data structure that works like a... stack
  - e.g. a stack of paper
  - Motivating example: call stack

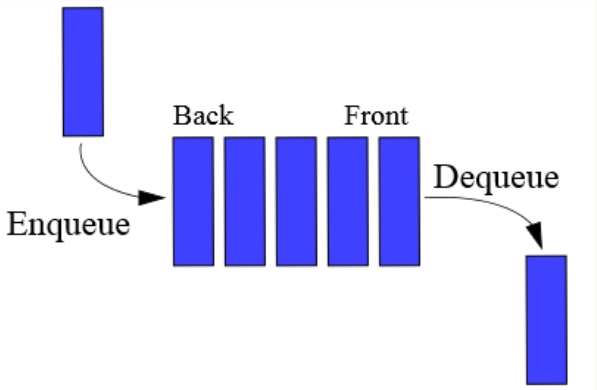



```
… void main(…) {
    a();
    b();
}

void b() {
    c();
    print(2);
}

void a() {
    b();
    print(1);
}

void c() {
    print(3);
}
```

Image Source: http://en.wikipedia.org/wiki/File:Data_stack.svg

Stacks and Queues

- Queue
  - Data structure that works like a... queue
  - or a "line" if you aren't British
  - Motivating example: printer queue



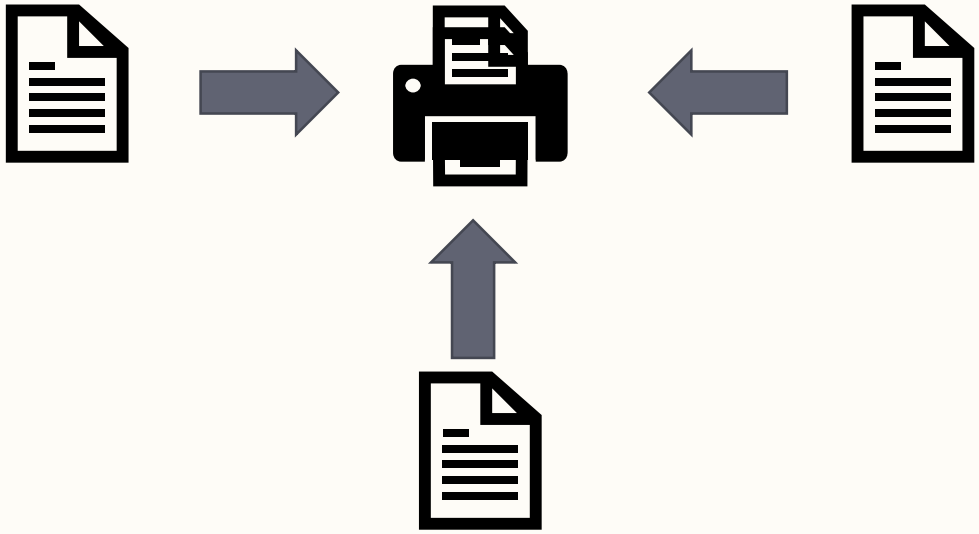


Image Source: http://en.wikipedia.org/wiki/File:Data_Queue.svg

Whiteboard…

Slides just here for reference!

Stacks White Board

- Push, pop, peek, isEmpty, size

- Arrays and/or Dynamic Arrays
  - Typical solution, good constants
- Linked List
  - Can use single linked list
  - Larger constant, but consistent performance

Queues White Board

- Enqueue, dequeue, peek, isEmpty, size

- Array w/size (or DynamicArray)

  + “Circular Queue”

- Linked List
  - Easy implementation with singly linked list+head/tail reference

Big-O?

- Typical to implement Stack using an Array or Dynamic List (good constraints)
- Typical to implement Queue using Singly Linked List

Stack

Operation Implementation	push	pop	peek	isEmpty	size
Dynamic Array					
Linked List					

Queue

Operation Implementation	enqueue	dequeue	peek	isEmpty	size
Dynamic Array					
Linked List					

* Amortized

Stacks and Queues Summary

- Typical to implement Stack using an Array or Dynamic List (good constraints)
- Typical to implement Queue using Singly Linked List

Stack

Operation Implementation	push	pop	peek	isEmpty	size
Dynamic Array	1*	1	1	1	1
Linked List	1	1	1	1	1

Queue

Operation Implementation	enqueue	dequeue	peek	isEmpty	size
Dynamic Array	1*	1	1	1	1
Linked List	1	1	1	1	1

* Amortized

Why the Restricted Operations?

- Why? Because good worst cases
  - O(1) for all supported operations
  - O(n) space
- Simple data structures
  - focus on limited operations
  - can be made out of primitive data structures (arrays and linked lists)
- Good for representing time-related data
  - call stack
  - packet queues

Project 2 Coming Friday!