

# CSE 332: Data Structures & Parallelism

Ruth Anderson

Winter 2021

Lecture 1

# Welcome!

---

We have 10 weeks to learn *fundamental data structures and algorithms for organizing and processing information*

- › “Classic” data structures / algorithms and how to analyze rigorously their efficiency and when to use them
- › Queues, dictionaries, graphs, sorting, etc.
- › Parallelism and concurrency (!)

# Today's Outline

---

- **Introductions**
- Administrative Info
- What is this course about?
- Review: Queues and stacks

# CSE 332 Course Staff!!

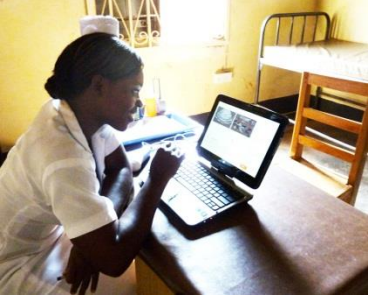
---

## **Instructor:**

Ruth Anderson

## **Teaching Assistants:**

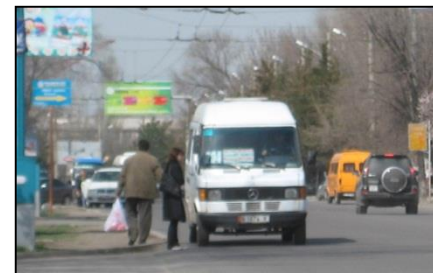
- Aashna Sheth
- Aayushi Modi
- Allen Aby
- Corinne Herzog
- Diya Joy
- Hamsa Shankar
- Harshal Desai
- Jeffery Tian
- Jevin Gu
- Kevin Pham
- Kris Wong
- Louis Maliyam
- Maggie Jiang
- Monty Nitschke
- Nachiket Karmarkar
- Richard Jiang
- Sashu Shankar
- Winston Jodjana



# Me (Ruth Anderson)

---

- **Grad Student at UW** in Programming Languages, Compilers, Parallel Computing
- **Taught Computer Science** at the University of Virginia for 5 years
- **Grad Student at UW**: PhD in Educational Technology, Pen Computing
- **Current Research**: Computing and the Developing World, Computer Science Education
- **Recently Taught**: data structures, architecture, compilers, programming languages, 142 & 143, data programming in Python, Unix Tools, Designing Technology for Resource-Constrained Environments



# Today's Outline

---

- Introductions
- **Administrative Info**
- What is this course about?
- Review: Queues and stacks

# Course Information

---

- **Instructor:** Ruth Anderson, CSE 558  
Office Hours: see course web page, and by appointment, ([rea@cs.washington.edu](mailto:rea@cs.washington.edu))
- **Text:** *Data Structures & Algorithm Analysis in Java*, (Mark Allen Weiss), 3rd edition, 2012  
(2<sup>nd</sup> edition also o.k.)
- **Course Web page:**  
<http://www.cs.washington.edu/332>

# Communication

---

- Course email lists:  
`cse332a_wi21@uw` or `cse332b_wi21@uw`
  - › You are already subscribed
  - › You must get and read announcements sent there
- Ed STEM Discussion board
  - › Your first stop for questions about course content & assignments
- Anonymous feedback link
  - › For good and bad: if you don't tell me, I won't know!



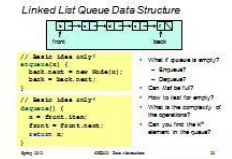
# Course Meetings

---

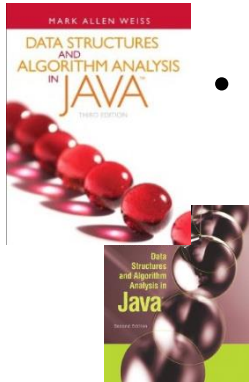
- Lecture
  - › Materials posted (sometimes afterwards), but take notes
  - › Ask questions, focus on key ideas (rarely coding details)
- Section
  - › Practice problems!
  - › Answer Java/project/homework questions, etc.
  - › Occasionally may introduce new material
  - › An important part of the course (not optional)
- Office hours
  - › Use them: *please visit us!*

# Course Materials

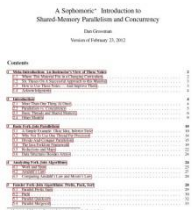
- Lecture and section materials will be posted
  - › But they are visual aids, not always a complete description!
  - › If you have to miss, find out what you missed



- Textbook: Weiss 3<sup>rd</sup> Edition in Java
  - › Good read, but only responsible for lecture/section/hw topics
  - › 3<sup>rd</sup> edition improves on 2<sup>nd</sup>, but we'll also support the 2<sup>nd</sup>



- Parallelism / concurrency units in separate free resources designed for 332



# Course Work

---

- ~20 Weekly individual homework exercises (25%)
- 3 programming projects (with phases) (40%)
  - › Use Java and IntelliJ, Gitlab
  - › Done in partners, o.k. if partner is in other quiz section
- No midterm or final exam!!! (30%)
  - › Instead, we will have 3 quizzes
  - › Quiz 1 & Quiz 2: Released on Wednesday, due on Friday
  - › Quiz 3: During Final Exam week
  - › Open book, small-group collaboration allowed
  - › *More details announced as we get closer to 1<sup>st</sup> quiz*
- Participation (5%) – available asynchronously

# Homework for Today!!

---

- 0) Project #1:** Fill out partner request survey(s) by 5pm TOMORROW
  - Optional Partner Mixers:  
Mon 7-8pm, Tues 11:30-12:30, 2:30-3:30pm
- 1) Review Java & install IntelliJ**
- 2) Exercise #1 – Due FRIDAY at 11:59pm**
- 3) Preliminary Surveys:** fill out by Wednesday evening
- 4) Reading in Weiss (see Syllabus)**

# Reading

---

- Reading in *Data Structures and Algorithm Analysis in Java*, 3<sup>rd</sup> Ed., 2012 by Weiss
- For this week:
  - › (Topic for Project #1) Weiss 3.1-3.7 – Lists, Stacks, & Queues
  - › (Wed) Weiss 2.1-2.4 –Algorithm Analysis
  - › (Useful) Weiss 1.1-1.6 –Mathematics and Java (Not covered in lecture – READ THIS)

# Today's Outline

---

- Introductions
- Administrative Info
- **What is this course about?**
- Review: Queues and stacks

# Data Structures + Parallelism

---

- About 70% of the course is a “classic data-structures course”
  - › Timeless, essential stuff
  - › Core data structures and algorithms that underlie most software
  - › How to analyze algorithms
- Plus a serious first treatment of programming with *multiple threads*
  - › For *parallelism*: Use multiple processors to finish sooner
  - › For *concurrency*: Correct access to shared resources
  - › Will make many connections to the classic material

# What 332 is about

---

- Deeply understand the basic structures used in all software
  - › Understand the data structures and their trade-offs
  - › Rigorously analyze the algorithms that use them (math!)
  - › Learn how to pick “the right thing for the job”
- Experience the purposes and headaches of multithreading
- Practice design, analysis, and implementation
  - › The elegant interplay of “theory” and “engineering” at the core of computer science



# Goals

---

- You will understand:
  - › what the tools are for storing and processing common data types
  - › which tools are appropriate for which need
- So that you will be able to:
  - › **make good design choices** as a developer, project manager, or system customer
  - › **justify** and **communicate** your design decisions

# One view on this course

---

- This is the class where you begin to think like a computer scientist
  - › You stop thinking in Java code
  - › You start thinking that this is a hashtable problem, a stack problem, etc.

# Data Structures?

---

“Clever” ways to organize information in order to enable *efficient* computation over that information.

# Example Trade-Offs

---

# Trade-Offs

---

A data structure strives to provide many useful, efficient operations

But there are unavoidable trade-offs:

- › Time vs. space
- › One operation more efficient if another less efficient
- › Generality vs. simplicity vs. performance

That is why there are many data structures and educated CSEers internalize their main trade-offs and techniques

- › And recognize logarithmic < linear < quadratic < exponential

# Getting Serious: Terminology

---

- Abstract Data Type (ADT)
  - › Mathematical description of a “thing” with set of operations on that “thing”
- Algorithm
  - › A high level, language-independent description of a step-by-step process
- Data structure
  - › A specific *organization of data* and family of algorithms for implementing an ADT
- Implementation of a data structure
  - › A specific implementation in a specific language

# The Stack ADT

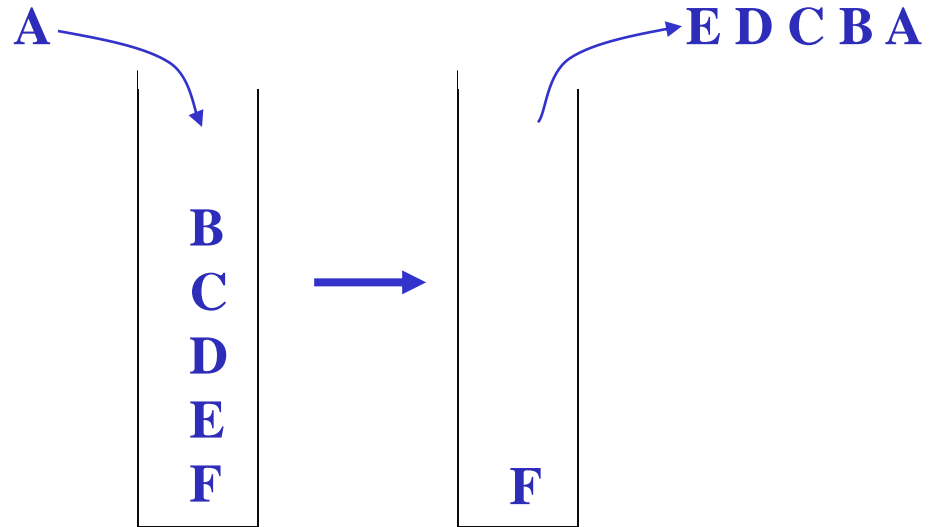
- Stack Operations:

`push`

`pop`

`top/peek`

`is_empty`



# Terminology Example: Stacks

---

- The **Stack ADT** supports operations:
  - › **push**: adds an item
  - › **pop**: raises an error if isEmpty, else returns *most-recently pushed item* not yet returned by a pop
  - › **isEmpty**: initially true, later true if there have been same number of pops as pushes
  - › ... (Often some more operations)
- A Stack **data structure** could use a linked-list or an array or something else, and associated **algorithms** for the operations
- One **implementation** is in the library `java.util.Stack`



# Why useful

---

The **Stack ADT** is a useful abstraction because:

- It arises **all the time** in programming (see Weiss for more)
  - › Recursive function calls
  - › Balancing symbols (parentheses)
  - › Evaluating postfix notation:  $3\ 4\ +\ 5\ *$
  - › Clever: Infix  $((3+4) * 5)$  to postfix conversion (see Weiss)
- We can code up a **reusable library**
- We can **communicate** in high-level terms
  - › “Use a stack and push numbers, popping for operators...”
  - › Rather than, “create a linked list and add a node when...”

# Today's Outline

---

- Introductions
- Administrative Info
- What is this course about?
- **Review: Queues and stacks**

# The Queue ADT

---

Queue Operations:

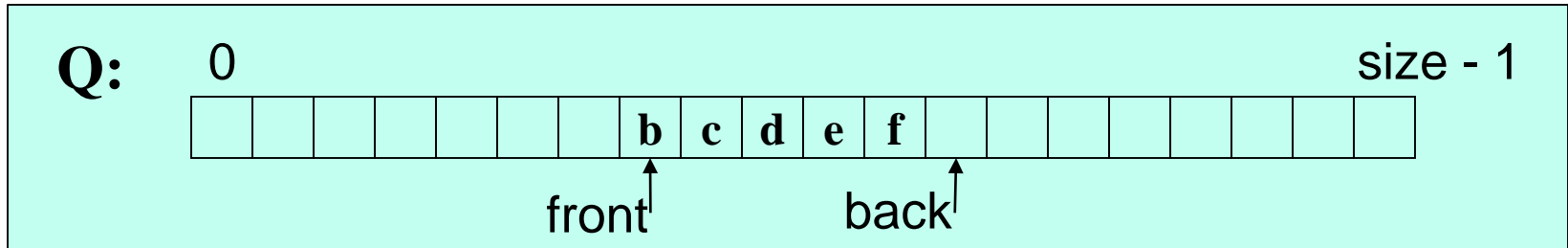
`enqueue`

`dequeue`

`is_empty`



# Circular Array Queue Data Structure

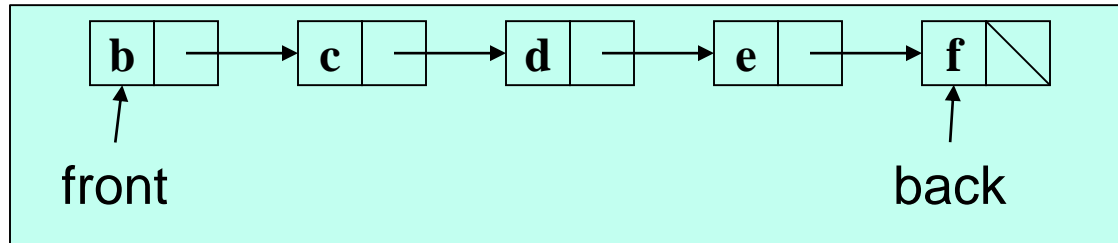


```
// Basic idea only!  
enqueue(x) {  
    Q[back] = x;  
    back = (back + 1) % size  
}
```

```
// Basic idea only!  
dequeue() {  
    x = Q[front];  
    front = (front + 1) % size;  
    return x;  
}
```

- What if **queue** is empty?
  - › Enqueue?
  - › Dequeue?
- What if **array** is full?
- How to *test* for empty?
- What is the *complexity* of the operations?

# Linked List Queue Data Structure



```
// Basic idea only!  
enqueue(x) {  
    back.next = new Node(x);  
    back = back.next;  
}
```

```
// Basic idea only!  
dequeue() {  
    x = front.item;  
    front = front.next;  
    return x;  
}
```

- What if **queue** is empty?
  - › Enqueue?
  - › Dequeue?
- Can **list** be full?
- How to *test* for empty?
- What is the *complexity* of the operations?

# Circular Array vs. Linked List

---

# Circular Array vs. Linked List

---

## Array:

- May waste unneeded space or run out of space
- Space per element excellent
- Operations very simple / fast

Operations not in Queue ADT, but also:

- Constant-time “access to  $k^{\text{th}}$  element”
- For operation “insertAtPosition”, must shift all later elements

## List:

- Always just enough space
- But more space per element
- Operations very simple / fast

Operations not in Queue ADT, but also:

- No constant-time “access to  $k^{\text{th}}$  element”
- For operation “insertAtPosition” must traverse all earlier elements

# Homework for Today!!

---

- 0) Project #1:** Fill out partner request survey(s) by 5pm TOMORROW
  - Optional Partner Mixers:  
Mon 7-8pm, Tues 11:30-12:30, 2:30-3:30pm
- 1) Review Java & install IntelliJ**
- 2) Exercise #1 – Due FRIDAY at 11:59pm**
- 3) Preliminary Surveys:** fill out by Wednesday evening
- 4) Reading in Weiss (see Syllabus)**