

Welcome to CSE 332!

CSE 332 Spring 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Annie Mao

Howard Xiao

Lucy Jiang

Chris Choi

Jade Watkins

Nathan Lipiarski

Ethan Knutson

Khushi Chaudhari

Richard Jiang

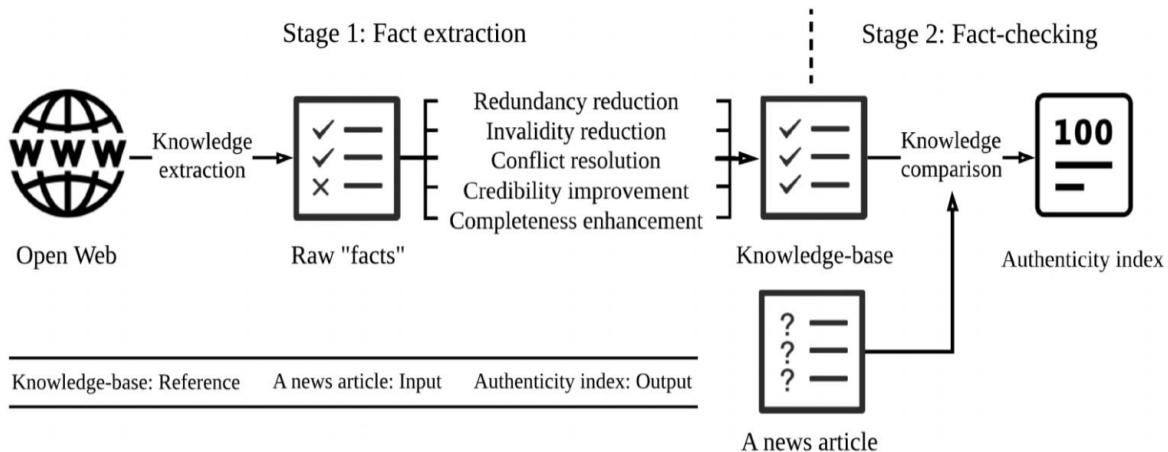
Lecture Outline

- ❖ **Introduction: Why This Course?**
- ❖ About This Course
 - Learning Objectives
 - People
 - Policies
- ❖ Abstract and Concrete Data Types
- ❖ List, Stack ,and Queue ADTs

Why: Increase Progress (?) in Society



Why: Discover New Knowledge



Why: Support Daily Life

How to search the internet

About 7,470,000,000 results (0.60 seconds)

Lecture Outline

- ❖ Introduction: Why This Course?

- ❖ About This Course
 - **Learning Objectives**
 - People
 - Policies

- ❖ Abstract and Concrete Data Types

- ❖ List, Stack, and Queue ADTs

Learning Objectives

- ❖ Learn fundamental *data structures* and *algorithms*
 - “Classic” data structures and algorithms
 - Queues, dictionaries, graphs, sorting, etc.
- ❖ Learn thought processes/patterns for *organizing* and *processing information*
 - Understand how to analyze their efficiency
 - Learn how to analyze tradeoffs and pick “the right tool for the job”
 - Parallelism and concurrency (!)
- ❖ This isn’t a “how to program” or “software engineering” class!
 - We will *practice* design, analysis, and implementation
 - Witness elegant interplay of “theory” and “engineering” at the core of computer science

Course Content

- ❖ What do we mean by “Data Structures and 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
- ❖ About 30% is programming with *multiple threads*
 - *Parallelism*: Use multiple processors to finish sooner
 - *Concurrency*: Correct access to shared resources
 - Will make many connections to the classic material

In other words ...

- ❖ 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, a sorting problem, etc.
 - You recognize tradeoffs
 - Time vs. space
 - One operation more efficient if another less efficient
 - Generality vs. simplicity vs. performance

- ❖ That is why we cover many data structures; educated CSEers internalize their tradeoffs and techniques
 - And recognize logarithmic < linear < quadratic < exponential

Lecture Outline

- ❖ Introduction: Why This Course?

- ❖ About This Course
 - Learning Objectives
 - **People**
 - Policies

- ❖ Abstract and Concrete Data Types

- ❖ List, Stack, and Queue ADTs

Introductions: Course Staff

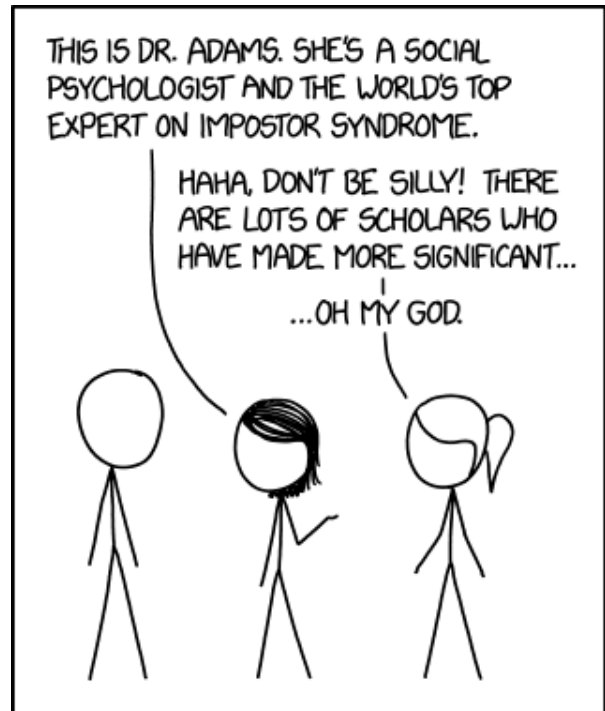
- ❖ Hannah C. Tang
 - UW CSE alumna with 17 years of bugs in industry

- ❖ TAs:
 - Annie Mao, Chris Choi, Ethan Knutson, Howard Xiao, Lucy Jiang, Jade Watkins, Khushi Chaudhari, Nathan Lipiarski
 - Consulting TA: Richard Jiang
 - Available in section, office hours, and discussion group
 - An invaluable source of information and help (!!)

- ❖ Get to know us
 - We are excited to help you succeed!
 - Schedule time for a virtual one-on-one to discuss anything

Introduction: Students

- ❖ ~120 students registered, scattered all around the world
 - When we're online only, it's easy to feel lost, as if everyone is "better" than you
- ❖ "Nearly 70% of individuals will experience signs and symptoms of impostor phenomenon at least once in their life."
 - https://en.wikipedia.org/wiki/Impostor_syndrome
- ❖ Our course size can be an asset!



<https://xkcd.com/1954>

Lecture Outline

- ❖ Introduction: Why This Course?

- ❖ About This Course
 - Learning Objectives
 - People
 - **Policies**

- ❖ Abstract and Concrete Data Types

- ❖ List, Stack, and Queue ADTs

Communication

- ❖ **Website:** <http://cs.uw.edu/332>
 - Schedule, policies, materials, assignments, etc.
- ❖ **Discussion:** <http://piazza.com/washington/spring2020/cse332>
 - Announcements made here
 - Ask and answer questions – staff will monitor and contribute
- ❖ **Office hours:** spread throughout the week
 - Can e-mail/private Piazza post to make individual appointments
- ❖ **Feedback:**
 - Anonymous feedback goes to Hannah, but she can't respond directly
 - `cse332-staff@cs` goes to the entire staff

Course Components

❖ Lectures

- Introduces the concepts (but rarely covers coding details)
- Please take notes!!! Slides posted after class
- (Hopefully) recorded; if Zoom password is necessary, use **cse332**

❖ Sections

- Practice problems and concept application
- Review materials (occasionally introduces new materials)
- Answer Java/project/homework questions

❖ Office Hours

- Use them!

Materials

- ❖ Textbook:
 - *Data Structures & Algorithm Analysis in Java*, Mark Allen Weiss
 - 3rd edition, 2012 (but 2nd edition ok)

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

Evaluation

- ❖ ~15 *individual* homework exercises (25%)

- ❖ 3 *partner-based* multi-phase programming projects (35%)
 - Use Java 11 and IntelliJ, Gitlab
 - Done in partners

- ❖ No midterm or final exam!!! (40%)
 - Instead, we will have 5 bi-weekly quizzes
 - Released on Wednesday, due on Thursday
 - Open book, small-group collaboration allowed. But no staff support (eg, Piazza or office hours questions)

Deadlines and Student Conduct

- ❖ Late policies
 - Exercises: no late submissions accepted
 - Projects: 3 “tokens” for the entire quarter
 - Every token is worth 1 late day

- ❖ Academic Conduct (**read** the full policy on the web)
 - In short: don’t attempt to gain credit for something you didn’t do and don’t help others do so either
 - This does **not** mean suffer in silence!
 - Learn from the course staff and peers, talk, share ideas; *but* don’t share or copy work that is supposed to be yours
 - Collaboration is **strongly** encouraged! Discuss confusing points with each other, because organizing your thoughts is the best way to learn!

Lecture Outline

- ❖ Introduction: Why This Course?

- ❖ About This Course
 - Learning Objectives
 - People
 - Policies

- ❖ **Abstract and Concrete Data Types**

- ❖ List, Stack, and Queue ADTs

Terminology: Data Structures vs Algorithms

❖ Data Structures:

- A way of organizing, storing, accessing, and updating a set of data
- *Examples from 14X*: arrays, linked lists, stacks, queues, trees

❖ Algorithms:

- A series of precise instructions guaranteed to produce a certain answer
- *Examples from 14X*: binary search, merge sort, recursive backtracking

Terminology: ADTs vs Concrete Data Structures

❖ Abstract Data Types (ADTs)

- Mathematical description of a “thing” and its set of operations

❖ Data Structures

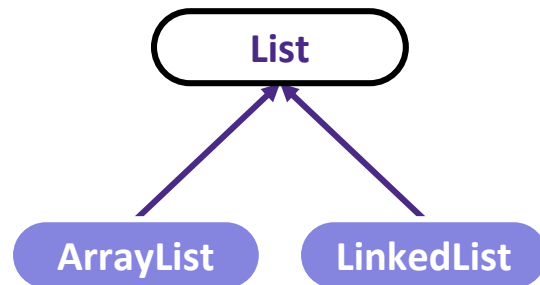
- *A way of organizing, storing, accessing, and updating a set of data*

❖ Implementations

- An implementation of an ADT is a data structure
- An implementation of a data structure are the collection of methods and variables in a specific language

A Helpful SAT-style analogy?

- ❖ In Java, an **interface** is a data type that specifies what to do but not how to do it
 - **List**: an ordered sequence of elements.
- ❖ A **subtype** implements all methods required by the interface
 - **ArrayList**: Resizable array implementation of the List interface
 - **LinkedList**: Doubly-linked implementation of the List interface



A Java interface is to a Java subtype, as an ADT is to a data structure!

Lecture Outline

- ❖ Introduction: Why This 'Course?
- ❖ About This Course
 - Learning Objectives
 - People
 - Policies
- ❖ Abstract and Concrete Data Types
- ❖ **List, Stack, and Queue ADTs**

List Functionality

List ADT. A collection storing an ordered sequence of elements.

- Each element is accessible by a zero-based index
- A list has a size defined as the number of elements in the list
- Elements can be added to the front, back, *or any index in the list*
- Optionally, elements can be removed from the front, back, *or any index in the list*

❖ Possible Implementations:

- ArrayList
- LinkedList

List Performance Tradeoffs

	ArrayList	LinkedList
addFront	linear	constant
removeFront	linear	constant
addBack	constant*	linear
removeBack	constant	linear
get(idx)	const	linear
put(idx)	linear	linear

* constant for most invocations

Stack and Queue ADTs

Stack ADT. A collection storing an ordered sequence of elements.

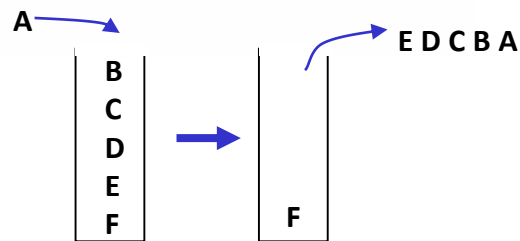
- A stack has a size defined as the number of elements in the stack
- Elements can only be added and removed from the top (“LIFO”)

Queue ADT. A collection storing an ordered sequence of elements.

- A queue has a size defined as the number of elements in the queue
- Elements can only be added to one end and removed from the other (“FIFO”)

Stack ADT

- ❖ **Stack**: an ADT representing an ordered sequence of elements whose elements can only be added/removed from one end.
 - Corollary: has “last in, first out” semantics (LIFO)
 - The end of the stack that we operate on is called the “top”
 - Operations:
 - `void push(Item i)`
 - `Item pop()`
 - `Item top() / peek()`
 - `boolean isEmpty()`
 - *(notably, there is no generic `get()` method)*



Terminology Example: Stacks

- ❖ The Stack **ADT** has the following operations:
 - **push**: adds an item
 - **pop**: raises an error if `isEmpty()`, else **removes** and **returns** *most-recently pushed item* not yet returned by a `pop()`
 - **top** or **peek**: same as `pop`, but doesn't remove the item
 - **isEmpty**: initially true, later true if there have been same number of `pop()`'s as `push()`'es
- ❖ A Stack **data structure** could use a linked-list or an array or something else. There are associated **algorithms** for each operation
- ❖ One **implementation** is in the library `java.util.Stack`

Stack Applications

- ❖ 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 **communicate** in shorthand and high-level terms
 - “Use a stack and push numbers”
 - Rather than: “create a linked list and add a node when you see a ...”

Stack Data Structure: Circular Array

❖ *State*

```
Item[] data;  
int size;
```

❖ *Behavior*

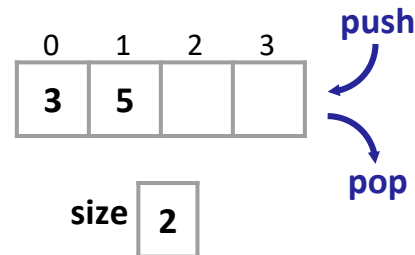
■ `push()`

- Resize data array if necessary
- Assign `data[size] = item`
- Increment `size`
- *Note: this is `ArrayList.addBack()`*

■ `pop()`

- Return `data[size]`
- Decrement `size`
- *Note: this is `ArrayList.removeBack()`*

```
push(3);  
push(4);  
pop();  
push(5);
```



Stack Data Structure: Linked List

❖ State

Node top;

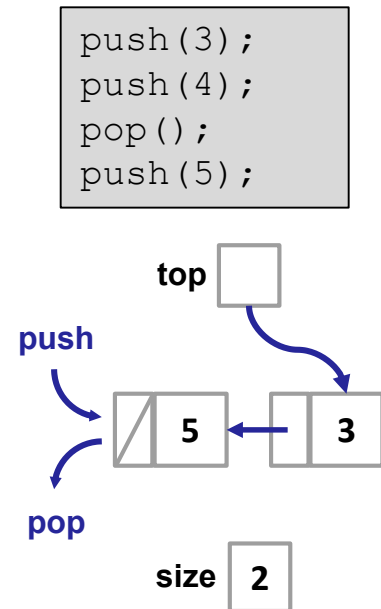
❖ Behavior

■ push ()

- Create a new node linked to top's current value
- Update top to new node
- Increment size
- *Note: this is `LinkedList.addBack()`*

■ pop ()

- Return top's item
- Update top
- Decrement size
- *Note: this is `LinkedList.removeBack()`*



Queue ADT

- ❖ **Queue**: an ADT representing an ordered sequence of elements, whose elements can only be added to one end and removed from the other end.
 - Corollary: has “first in, first out” semantics (FIFO)
 - Two methods:
 - `void enqueue(Item i)`
 - `Item dequeue()`
 - `boolean isEmpty()`
 - *(notably, there is no generic `get()` method)*



Queue Data Structure: Array

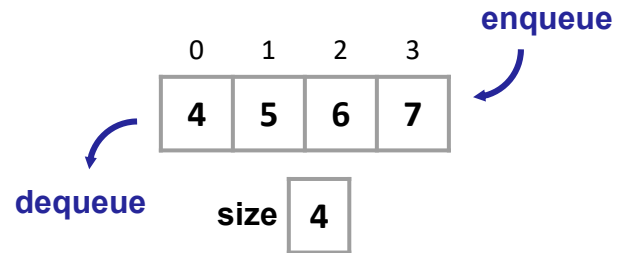
❖ *State*

```
Item[] data;  
int size;
```

❖ *Behavior*

- `enqueue ()`
 - `ArrayList.addBack ()`
- `dequeue ()`
 - `ArrayList.removeFront ()`

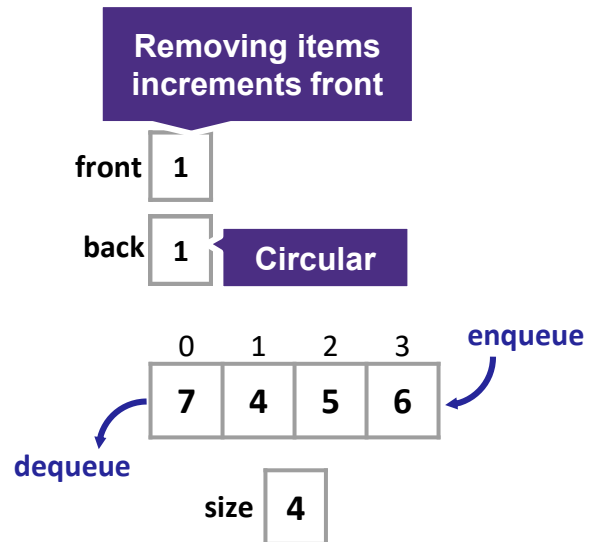
```
enqueue (3) ;  
enqueue (4) ;  
dequeue () ;  
enqueue (5) ;  
enqueue (6) ;  
enqueue (7) ;
```



Queue Data Structure: Circular Array

- ❖ The front of the queue does not need to be the front of the array!
 - This data structure is also known as a **circular array**

```
enqueue (3) ;  
enqueue (4) ;  
dequeue () ;  
enqueue (5) ;  
enqueue (6) ;  
enqueue (7) ;
```



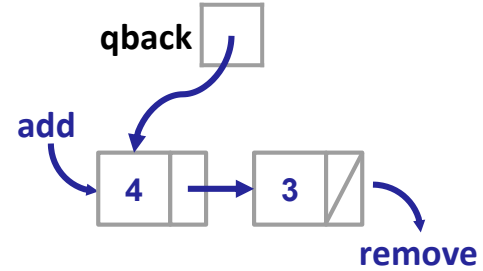
Queue Data Structure: Linked List

❖ *State*

```
Node qback; // front of list
            // is back of
            // queue
```

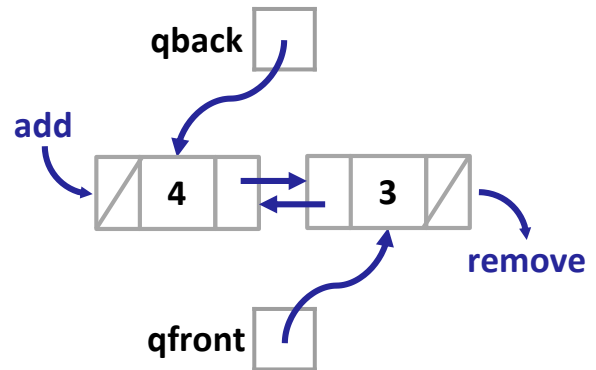
❖ *Behavior*

- enqueue ()
 - LinkedList.addLast ()
- dequeue ()
 - LinkedList.removeFront ()



Queue Data Structure: Doubly Linked List

- ❖ What if we made the list doubly-linked and added a **front** pointer?



Homework for Today!!

- ❖ Project #1:
 - Fill out partner request survey(s) by 6pm PDT TOMORROW

- ❖ Review Java & install IntelliJ

- ❖ Exercise #1
 - “Due” FRIDAY at 8:59pm

- ❖ Readings in Weiss
 - *By Wednesday*: Weiss 3.1-3.7 – Lists, Stacks, & Queues
 - *By Friday*: Weiss 2.1-2.4 – Algorithm Analysis
 - *Optional*: Weiss 1.1-1.6 – Mathematics and Java
 - Not covered in lecture

Summary (1 of 2)

❖ Data Structures:

- A way of organizing, storing, accessing, and updating a set of data

❖ Algorithms:

- A series of precise instructions guaranteed to produce a certain answer

❖ Abstract Data Types (ADTs)

- Mathematical description of a “thing” and its set of operations

❖ Implementations

- An implementation of an ADT is a data structure
- An implementation of a data structure are the collection of methods and variables in a specific language

Summary (2 of 2)

List ADT. A collection storing an ordered sequence of elements

- Each element is accessible by a zero-based index
- A list has a size defined as the number of elements in the list
- Elements can be added to the front, back, *or any index in the list*
- Optionally, elements can be removed from the front, back, *or any index in the list*

Stack ADT. A collection storing an ordered sequence of elements.

- A stack has a size defined as the number of elements in the stack
- Elements can only be added and removed from the top (“LIFO”)

Queue ADT. A collection storing an ordered sequence of elements.

- A queue has a size defined as the number of elements in the queue
- Elements can only be added to one end and removed from the other (“FIFO”)